UNIVERSIDADE FEDERAL DO PARANÁ
Mestrado em Informática

ALGORITMOS GENÉTICOS PARA SOLUÇÃO DE PROBLEMAS DE ALCANÇABILIDADE EM UMA DETERMINADA CLASSE DE REDES DE PETRI ACÍCLICAS

CÁSSIO SOARES CARVALHO
CURITIBA

2007

CÁSSIO SOARES CARVALHO

ALGORITMOS GENÉTICOS PARA SOLUÇÃO DE PROBLEMAS DE ALCANÇABILIDADE EM UMA DETERMINADA CLASSE DE REDES DE PETRI ACÍCLICAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em informática. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Castilho

CURITIBA

2007

CÁSSIO SOARES CARVALHO

ALGORITMOS GENÉTICOS PARA SOLUÇÃO DE PROBLEMAS DE ALCANÇABILIDADE EM UMA DETERMINADA CLASSE DE REDES DE PETRI ACÍCLICAS

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. Marcos Castilho
Departamento de Informática, UFPR

Profª. Drª. Leliane Nunes de Barros
IME-USP/SP

Prof. Dr. Fabiano Silva
Departamento de Informática, UFPR

26 de fevereiro de 2007

Curitiba

AGRADECIMENTOS

Inicialmente gostaria de agradecer ao meu orientador, Marcos Castilho, pela amizade, apoio e dedicação ao longo deste trabalho. Muito obrigado pelas oportunidades.

Gostaria também de expressar meus agradecimentos a todos os membros do Departamento de Informática da Universidade Federal do Paraná, especialmente ao Alexandre Direne, ao Fabiano Silva e à Laura García. Agradeço ainda à minha ex-professora de graduação, Silvia Botelho, da Fundação Universidade Federal do Rio Grande.

Agradeço aos meus pais pelo apoio incondicional e pela educação que deles recebi. À minha irmã Camila, minha afilhada Laura e minha "segunda mãe" Lenícia. Aos cunhados Arthur e Graciele pelo apoio no Estado do Paraná.

Não poderia deixar de mencionar também o nome de grandes amigos e colegas que participaram de minha formação acadêmica: Jean Orengo; Marcelo Linder; Gustavo Sabin.

Finalmente dedico este trabalho ao meu amor, Renata Cantarelli, e aos meus pais João Inácio Carvalho e Nádia Carvalho. Essa conquista é nossa. Amo muito vocês!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

Este trabalho apresenta uma abordagem baseada em algoritmos genéticos a qual permite solucionar conflitos em uma determinada classe de redes de Petri. A solução destes conflitos é uma solução para o problema de planejamento em inteligência artificial. Após ser apresentado o estado da arte na área, será feita uma análise de alguns sistemas os quais também são baseados em paradigmas evolutivos, comparando-os com as características do presente trabalho e que contribuiram para direcionar a pesquisa. Por fim, expõe-se tanto o algoritmo desenvolvido quanto os resultados obtidos.

ABSTRACT

This work presents an approach based on genetic algorithms which allows to solve conflicts in a specific Petri net class. The solution to this conflicts it's a solution to the planning problem in artificial intelligence. After to be presented the state of the art in the area, will be made an analysis of some systems which also are based on evolutionary paradigms, comparing it with the features of the present work and that had contributed direct the research. Finally, it's exposed the algorithm and it results.

CAPÍTULO 1

INTRODUÇÃO

Diariamente as pessoas executam diversas tarefas com o intuito de satisfazer suas necessidades. Muitas decisões são tomadas em questão de milésimos de segundos, enquanto outras porém, certamente exigem horas de muita atenção. Embora essas duas situações possam parecer distintas no sentido de complexidade ou de tempo de raciocínio exigido, ambas têm características fundamentais que caracterizam um problema de planejamento: uma descrição atual do ambiente envolvido, uma descrição desejada para o ambiente em algum momento no futuro e, por último, uma descrição das ações disponíveis para modificar um determinado ambiente. Traduzindo estes aspectos para o problema de planejamento clássico encontra-se a noção de estado inicial, estado objetivo e teoria do domínio, respectivamente.

O programa o qual permite descobrir como transitar entre o estado inicial e o estado objetivo a partir da teoria do domínio é conhecido como sistema planejador. A resposta de um planejador clássico é uma seqüência de ações, ou melhor, um plano [1][2].

A base atual na área de planejamento começou a ser consolidada em 1971, quando Fikes e Nilson apresentaram o sistema STRIPS[1] [3]. Tal sistema planejador teve como destaque a simplicidade utilizada na representação de estados e de ações, o que o fez ser muito lembrado por "representação STRIPS".

O surgimento do sistema STRIPS pode ser visto como um marco na história do planejamento em inteligência artificial, visto que o mesmo permitiu tratar os problemas dessa área de uma forma diferente. Problemas os quais, até então, eram solucionados com procedimentos de prova de teoremas e o uso de lógica clássica [4], passaram a ser tratados como um problema de busca em um espaço de estados. Tal feito possibilitou também evitar o problema de persistência (*frame problem*) [5][6].

[1]O acrônimo STRIPS faz referência a "STanford Research Insitute Problem Solver"

Por outro lado, o sucesso do sistema proposto esbarrou no tamanho do espaço de busca gerado, o qual era exageradamente grande inclusive para problemas muito simples. A partir de então, a área de planejamento em inteligência artificial, cuja complexidade é PSPACE-Completo [7][8], não recebeu nenhuma contribuição significativa durante duas décadas.

Nos anos 90 surgiram algumas alternativas de buscas no espaço de planos, resultando no planejador *UCPOP* [9], e também idéias de se reduzir um problema de planejamento em um problema SAT[2], quando surgiu o *SatPlan* [10]. Este último planejador, baseado em formulas SAT, obteve resultados bem satisfatórios para a época.

Provavelmente o maior avanço na área de planejamento tenha ocorrido em 1995, quando Blum e Furst apresentaram o *Graphplan* [11]. A contribuição do *Graphplan* foi construir um grafo o qual compacta a representação do espaço de busca, tendo como partida a representação STRIPS. O algoritmo proposto é constituído de duas fases: a primeira responsável pela criação de um grafo de planos e uma segunda que realiza a busca por um plano neste grafo. O destaque da proposta estava presente na primeira fase, a qual continha a otimização da representação do espaço de busca. Várias propostas, então, começaram a surgir utilizando a primeira fase do *Graphplan* e substituindo a segunda fase por uma abordagem alternativa.

Dentre os descendentes do *Graphplan* pode-se citar o *Blackbox* [12], desenvolvido por Kautz e Selman, e o *FF* [13], desenvolvido por Hoffmann e Nebel.

O *Blackbox* utiliza técnicas de satisfabilidade para solucionar problemas de planejamento. Na verdade, seu algoritmo é resultado de uma combinação entre o *Satplan* e o grafo de planos gerado pelo *Graphplan*. Tal combinação obteve muito sucesso, visto que as fórmulas SAT obtidas a partir de um grafo de planos são bem menores do que as até então consideradas pelo *SatPlan*. Já o *FF* se destaca pela combinação do grafo de planos com um método de subida de encosta clássico. Neste caso, uma variante do grafo de planos é usada como função heurística para guiar a busca local. Ambos planejadores são considerados os mais rápidos planejadores clássicos da atualidade.

[2]Boolean Satisfiability Problem.

Recentemente, o *PetriPlan* [14][15], proposto por Fabiano Silva e colegas, apresentou uma nova abordagem para tratar problemas de planejamento. Sua proposta consiste em traduzir um problema de planejamento em inteligência artificial para um problema de alcançabilidade em redes de Petri acíclicas [16].

A idéia do *PetriPlan* em utilizar uma representação baseada em redes de Petri contribui muito para um melhor entendimento de como as ações de um problema se relacionam, permitindo uma melhor análise dos conflitos existentes no espaço de busca do grafo de planos. Outro ponto muito importante é que a rede de Petri é um formalismo matemático o qual oferece um conjunto de equações, baseadas em cálculo matricial, muito úteis para análise da dinâmica de sistemas.

Razer Montaño [17], por exemplo, desenvolveu um planejador utilizando como base o *Petriplan*. Em seu trabalho, a rede de Petri é convertida em uma instância SAT em NNF (Negation Normal Form).

A computação evolutiva é, até então, uma abordagem pouco explorada na área de planejamento. Essa abordagem tem como princípio evoluir um conjunto de soluções iniciais (candidatas) até que, alguma delas, atinja um nível satisfatório de qualidade com relação a um determinado problema em análise. Para tal, leva-se em consideração a teoria da seleção natural proposta por Charles Darwin [18].

No contexto dessa área pode-se citar o uso de um de seus principais paradigmas, os algoritmos genéticos [19], em alguns casos como: Lecheta [20], com a finalidade de solucionar um problema de planejamento, gera e evolui um conjunto de soluções candidatas a partir da representação STRIPS; Castilho e colegas [21], visando executar e otimizar o mesmo processo, utiliza um grafo de planos como referência para gerar o conjunto de soluções candidatas. Tal escolha permite a análise de uma região bem mais restrita do espaço de busca, resultando um planejador cujo desempenho é bem melhor com relação ao de Lecheta; Westerberg e Levine propuseram o "programação genética" [22], também com o propósito de tratar problemas de planejamento no domínio STRIPS.

O uso de algoritmos genéticos (AG) também alcançou resultados satisfatórios na busca por soluções em fórmulas SAT. É o caso do algoritmo *GASAT* [23][24], o qual ficou bem

colocado na competição internacional de 2004 (SAT'04 Competition).

O grupo do laboratório de inteligência computacional (LIC) do departamento de informática (DInf) da Universidade Federal do Paraná (UFPR) acredita que, para um bom funcionamento de um planejador baseado em AG, seja necessário analisar e definir uma relação de importância entre as ações ou conjuntos de ações disponíveis para constituir um plano. Tal característica permitiria formar estruturas cromossômicas onde se conhece melhor o poder de influência de cada ação sobre o objetivo.

Tendo isso em vista, e verificando melhor a representação utilizada pelo *PetriPlan*, percebemos que a classe de redes de Petri gerada forma uma base muito acessível para desenvolver um novo planejador baseado em algoritmos genéticos. Sendo assim, este trabalho propõe solucionar conflitos nessas redes de Petri a fim de solucionar um problema de alcançabilidade e, conseqüentemente, encontrar um plano para o referente problema de planejamento clássico.

Os próximos capítulos estão organizados da seguinte forma. Enquanto no capítulo 2 é realizada uma revisão do estado da arte na área de planejamento, no capítulo 3 é descrito o paradigma dos algoritmos genéticos. O capítulo 4 expõe como o planejamento vem sendo tratado até então com o uso de AG e, na seqüencia, é proposto um planejador (*GAPNet*) baseado no mesmo paradigma com intuito de resolver um problema de alcançabilidade em redes de Petri acíclicas. Ainda neste capítulo, são apresentados os resultados obtidos por esse sistema. O capítulo 5 analisa a contribuição deste trabalho para a área em que o mesmo se insere, indicando aspectos os quais podem ser relevantes em futuras pesquisas.

CAPÍTULO 2

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Este capítulo contém uma base teórica sobre planejamento clássico baseado na representação STRIPS, descrevendo também o ambiente de planejamento Ipê (IPE), a representação baseada em redes de Petri utilizada pelo *Petriplan* e o paradigma dos algoritmos genéticos. Suas seções estão organizadas da seguinte forma. A seção 2.1 define os conceitos sobre planejamento baseado em STRIPS, a seção 2.2 apresenta o sistema planejador *Graphplan*, a seção 2.3 diz respeito à linguagem de definição de domínios (PDDL), a seção 2.4 contém uma descrição do IPE, a seção 2.5.1 apresenta a abordagem baseada em redes de Petri utilizada pelo planejador *Petriplan* e, por fim, a seção 3 expõe o paradigma dos algoritmos genéticos.

2.1 Conceitos

No planejamento clássico em inteligência artificial um problema é caracterizado por três informações:

- uma descrição de um estado inicial;
- uma descrição do estado objetivo;
- uma descrição de possíveis ações a serem executadas (teoria do domínio).

Formalmente, pode-se definir um problema de planejamento como sendo a tripla $P =< O, I, G >$. Onde I é o estado inicial, G é o estado objetivo e O é o conjunto de ações existentes para formar um plano.

Sendo assim, o objetivo de um sistema planejador é encontrar um conjunto de ações que, ao serem executadas em determinada ordem, transformam o estado inicial no estado objetivo. Para que isso seja possível, um planejador deve considerar as ações contidas na teoria do domínio, definida pelo conjunto O.

Uma vez que é necessário transitar entre diferentes estados a partir de um conjunto de ações, a forma com que essas informações são descritas vem a ser um ponto muito importante. Sendo assim, justamente por sua simplicidade, a representação STRIPS se tornou a base do planejamento em inteligência artificial. Sua contribuição, mais especificamente, consiste em permitir que problemas dessa área sejam tratados de forma diferente. Problemas que antes eram solucionados com procedimentos de prova de teoremas e o uso de lógica clássica [4], passaram a ser tratados como um problema de busca em um espaço de estados. Tal feito possibilitou também evitar o problema de persistência (*frame problem*) [5][6].

Na representação STRIPS, um estado é modelado como um conjunto de literais instanciados. Por exemplo, considere a seguinte situação:

- a televisão está ligada;
- o controle remoto está sem pilhas.

Isto poderia ser representado em STRIPS da seguinte forma:

$$\textit{ligada(televisão)} \wedge \textit{sempilhas(controleremoto)}$$

Genericamente, para um estado com n características, tem-se o seguinte:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_{n-1} \wedge C_n$$

Repare que, na definição de um estado, é utilizado o operador "$\wedge$" para representar uma conjunção de literais (características). Isso significa que tal estado existe se e somente se todos os literais por ele indicados também existirem.

De acordo com a hipótese do mundo fechado [25], tudo aquilo que não estiver declarado em um estado é assumido como falso. No exemplo mostrado a cima, percebe-se que existem apenas duas características verdadeiras. Sabendo-se que existe uma característica a qual determina a disponibilidade do controle remoto, conclui-se que o mesmo não se encontra disponível, visto a ausência de uma declaração *"disponível(controleremoto)"*. Ou seja, não há necessidade de incluir uma declaração do tipo *"¬ disponível(controleremoto)"*.

A definição de uma ação segundo a representação STRIPS é mais complexa, sendo necessário três informações:

- uma lista (*pre*) de pré-condições para que a ação seja executada;
- uma lista (*pos*) de características a serem adicionadas após a execução da ação;
- e uma lista (*del*) de características a serem removidas após a execução da ação.

Formalmente, pode-se definir uma ação como sendo a tripla:

$$A_0 = (pre(A_0), pos(A_0), del(A_0))) \tag{2.1}$$

Define-se que uma ação está habilitada para ser executada se e somente se as suas pré condições estiverem presentes na declaração do estado atual. Posteriomente à execução de uma determinada ação, os literais do estado atual devem ser atualizados de acordo com as remoções e adições listadas na descrição da mesma.

Suponha a existência de uma ação A que tenha a seguinte representação STRIPS:

- Descrição: Coloca pilhas no controle remoto;
- Lista pre: *sempilhas(controleremoto)*, *disponível(controleremoto)*, *disponível(pilhas)*
- Lista pos: *compilhas(controleremoto)*
- Lista del: *disponível(pilhas)*, *sempilhas(controleremoto)*

Agora, dado um estado t_0 representado por:

$$\textit{ligada(televisão)} \wedge \textit{sempilhas(controleremoto)} \wedge \textit{disponível(controleremoto)} \wedge \textit{disponível(pilhas)}$$

A execução de A no momento em que o estado atual é t_0, resulta em uma transição para o estado t_1 descrito por:

$$\textit{ligada(televisão)} \wedge \textit{compilhas(controleremoto)} \wedge \textit{disponível(controleremoto)}$$

Essa simplicidade da representação STRIPS permitiu que a mesma se tornasse uma referência na área de planejamento em inteligência artificial. Por outro lado, o enorme espaço de busca por ela gerado, inclusive para problemas muito pequenos, vem a ser uma desvantagem.

Após o surgimento da representação STRIPS, passaram-se duas décadas sem que nenhuma contribuição significativa ocorresse na área. Foi então que, em 1995, Blum e Furst propuseram o que foi provavelmente um do maiores passos na área de planejamento: o *Graphplan*.

2.2 O *Graphplan*

O *Graphplan* [11], apresentado por Blum e Furst em 1995, foi responsável por um dos maiores avanços na área de planejamento. Seu algoritmo é constituído de duas fases: expansão e busca. Sua grande contribuição está presente na primeira fase do algoritmo, a qual permite gerar em tempo polinomial uma representação mais compacta para o espaço de busca. Tal procedimento permitiu tratar uma variedade bem maior de problemas, servindo de motivação para os pesquisadores da área. A seguir, serão descritas as duas fases que constituem o *Graphplan*.

2.2.1 Expansão

Nesta primeira fase o algoritmo gera um grafo organizado em camadas, o qual é conhecido como grafo de planos. Esse grafo possui dois tipos de nodos: nodos literais, que representam características do estado do mundo, e nodos ações, que representam ações. Sua estrutura alterna entre camadas constituídas apenas por literais e camadas constituídas apenas por ações.

As arestas que conectam os nodos entre diferentes camadas podem representar tanto uma pré-condição como um efeito de uma determinada ação. Sendo assim, quando uma aresta conecta um nodo literal a um nodo ação, significa que este literal é uma pré-condição desta ação. Quando uma aresta conecta um nodo ação a um nodo literal, significa que

este literal é um efeito desta ação.

Em muitos casos, ações contidas em uma mesma camada não poderão ser executadas concomitantemente visto a existência de algum conflito entre as mesmas. Neste caso, insere-se um arco ligando as ações conflitantes com o intuito de indicar uma relação de exclusão mútua. Esses conflitos são propagados também para os nodos literais, denotando que tais características não podem existir ao mesmo tempo. Sendo assim, os literais presentes em uma determinada camada, associados às relações de exclusão mútua da mesma, constituem um conjunto de possíveis estados.

A figura 2.1, gerada a partir da classe grafo de planos do ambiente IPE (seção 2.4), mostra a estrutura de um grafo de planos que modela o problema *probex* descrito no anexo I. Nele, pode-se perceber a existência de 5 camadas, onde A_1 e A_3 são camadas de ações e P_0, P_2 e P_4 são de literais. Os identificadores numéricos presentes em cada nodo são uma referência para a respectiva ação ou para o respectivo literal representado. No caso dos literais, a existência de um identificador negativo indica que o mesmo representa a negação de uma característica. Por exemplo, se (1) representa A, logo (-1) representa $\neg A$.

Além das ações descritas no domínio em questão, o grafo de planos inclui também ações denominadas de manutenção. Essas ações têm por finalidade, unicamente, manter de um estado para outro aquelas características que não foram modificadas por nenhuma das ações executadas. Uma ação de manutenção tem um mesmo literal como pré-condição e efeito, como por exemplo, a ação 35 da camada A_1 da figura 2.1.

Antes de descrever como se dá o processo de expansão do grafo de planos, será analisado em quais situações deve-se incluir uma relação de exclusão mútua no grafo. Tais relações podem existir tanto entre ações quanto entre literais.

Dado duas ações a_1 e a_2 presentes em um mesma camada i do grafo de planos, as mesmas são consideradas mutuamente exclusivas quando ocorrer pelo menos um dos casos abaixo descritos:

- Efeitos inconsistentes: o efeito de uma ação é a negação de um dos efeitos da outra ação;

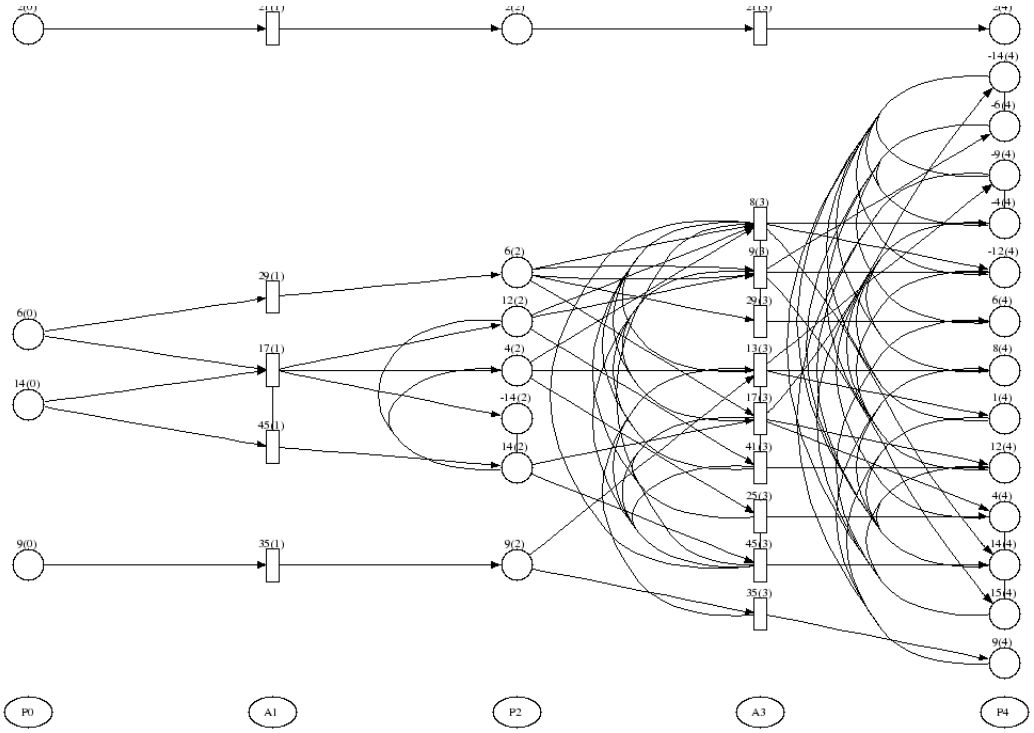

Figura 2.1: Grafo de planos - Estrutura

- Interferência: um efeito de uma ação é a negação de uma pré-condição da outra ação;

- Competição de necessidades: as ações a_1 e a_2 têm pré-condições as quais são mutuamente exclusivas na camada $i-1$.

Da mesma forma, dado dois literais p_1 e p_2 presentes em uma mesma camada j do grafo de planos, os mesmos são considerados mutuamente exclusivos quando ocorrer pelo menos um dos seguintes casos:

- um literal é a negação do outro;

- todas as maneiras de obter os literais p_1 e p_2 são mutuamente exclusivas entre si, ou seja, as ações da camada $j-1$ as quais têm como efeito os respectivos literais são duas a duas mutuamente exclusivas. Essa condição é denominada *suporte inconsistente*.

Uma vez definida as situações em que as relações de exclusão mútua devem ser indicadas, pode-se descrever como o grafo de planos deve ser gerado.

Inicialmente, o grafo de planos é composto por apenas uma camada (camada $n = 0$), a qual representa o estado inicial do problema que está sendo tratado. A expansão, então, consiste em incluir novas camadas de ações e de literais até que uma condição de parada seja satisfeita. Neste caso, a condição de parada engloba dois requisitos:

- encontrar, em uma mesma camada, todos os literais presentes no estado objetivo;
- os literais objetivos não podem conter conflitos entre si.

Essa condição de parada é dita ser necessária porém não suficiente, uma vez que as relações de exclusão mútua do grafo de planos não determinam todos os conflitos possíveis entre ações no problema considerado. Embora tal condição não garanta a existência de um plano no grafo gerado, deve-se passar para a próxima fase e realizar uma busca no mesmo. Caso não seja possível encontrar um plano nessa estrutura, deve-se voltar à fase de expansão e incluir mais uma camada de ações e outra de literais. Se a partir de determinado momento a fase de expansão não implicar mudanças no grafo de planos, significa que não existe solução para o problema em questão.

O algoritmo 1 descreve os passos executados durante a fase de expansão do grafo de planos.

Algoritmo 1 Expansão do grafo de planos

1: Gere uma camada (camada 0) contendo os literais presentes no estado inicial;
2: $n = 0$;
3: Gere uma camada $n + 1$ composta por todas ações que podem ser executadas em um dos possíveis estados definidos pela camada n de estados, ligando cada ação a suas respectivas pré condições;
4: Gere uma camada $n + 2$ composta por todos os literais(efeitos) das ações presentes na camada $n + 1$, ligando cada literal a respectiva ação que o gera;
5: Inclua as ligações necessárias, tanto entre literais quanto entre ações, para indicar a existência de conflitos;
6: $n = n + 2$;
7: Caso a camada n contenha todos os literais do estado objetivo e, ao mesmo tempo, não existam conflitos entre os mesmos, retorne a estrutura até então gerada;
8: Caso a camada n seja idêntica a camada $n - 2$, retorne "*Grafo estagnado*";
9: Volte ao passo 3;

2.2.2 Busca

A segunda fase do *Graphplan* consiste de uma busca para trás no grafo de planos gerado na primeira fase. Neste processo, o ponto crítico é a necessidade de tomar decisões sobre quais ações selecionar de acordo com os conflitos existentes.

No princípio, o algoritmo seleciona apenas os literais presentes na última camada e que também fazem parte do estado objetivo. A meta a partir de então é escolher, da camada anterior, ações as quais têm como efeito os literais do estado objetivo, respeitando, é claro, o conjunto de conflitos pertinentes a essa camada.

Uma vez efetuado este procedimento, os literais que são pré-condições das ações escolhidas passam a ser o novo objetivo a ser satisfeito. Sendo assim, deve-se repetir o procedimento de escolha de ações considerando agora a camada precedente destes novos literais. Tal processo deve ser executado até que todos os literais do conjunto de novos objetivos estejam presentes no estado inicial, caracterizando uma situação a qual indica que um plano foi encontrado.

O algoritmo 2 descreve de forma um pouco mais detalhada como se dá o processo de busca no grafo de planos.

Caso a busca do algoritmo 2 falhe, ou seja, caso não existam mais alternativas a serem testadas, significa que não existe um plano válido no grafo. Acontecendo isso, deve-se voltar a fase de expansão e gerar uma nova camada de ações e uma nova camada de literais. O surgimento de uma nova camada de ações pode viabilizar a execução de duas ações antes conflitantes, fazendo por exemplo com que as mesmas sejam executadas em série ao invés de em paralelo.

Além desta descrição apresentada para o grafo de planos, o mesmo possui também uma variação conhecida como grafo de planos relaxado. A diferença é que, neste caso, a fase de expansão descrita pelo algoritmo 1 desconsidera a existência de conflitos entre ações. Em outras palavras, significa dizer que o procedimento que gera o grafo de planos não considera a lista de remoção de literais das ações executadas. Sendo assim, não há como os efeitos de uma ação interferirem nos efeitos ou pré-condições de outra. Embora a versão relaxada não contenha uma descrição completa do espaço de busca, a estrutura

Algoritmo 2 Busca no grafo de planos

1: $n = (NumeroDeCamadas) - 1$;
2: Crie uma lista L contendo os literais da camada n que também estão presentes no estado objetivo;
3: Crie uma lista vazia A;
4: **for all** $l \mid l \in L$ **do**
5: Encontre e selecione uma ação a na camada $n - 1$ a qual contenha como efeito o próprio literal l;
6: **if** (ação a não é conflitante com as ações presentes em A) **then**
7: Insira a em A;
8: Crie uma lista E contendo os efeitos da ação a;
9: **for all** $e \mid e \in E$ **do**
10: **if** $e \in L$ **then**
11: Remova e de L;
12: **if** $L \neq \emptyset$ **then**
13: Retroceda e modifique as ações escolhidas até então no passo 4;
14: **for all** $a \mid a \in A$ **do**
15: Encontre os literais na camada $n - 2$ os quais satisfazem as pré-condições de a e insira-os em L;
16: $n = n - 2$;
17: **if** $(\forall l \in L, l \in EstadoInicial)$ **then**
18: Um plano foi encontrado.
19: **else**
20: Volte ao passo 3;

gerada neste caso exige menos tempo de processamento e fornece informações importantes que podem ser usadas como métricas em funções heurísticas de outros planejadores, como por exemplo o *FF* [13].

Portanto, o *Graphplan* propõe uma nova estrutura para tratar problemas de planejamento. Essa estrutura tem como diferencial a capacidade de representar o espaço de busca de forma mais compacta do que as representações até então existentes. Por outro lado, o algoritmo proposto não apresenta nenhuma heurística sofisticada para decidir sobre quais ações/literais selecionar na fase de busca.

Essa nova representação, o grafo de planos, despertou o interesse de pesquisadores da área e se tornou uma referência em planejamento. A medida que as pesquisas se intensificaram, o interesse de protagonizar disputas entre sistemas planejadores deu origem a Competição Internacional de Planejadores organizada pelo congresso AIPS[1] [26], e que atualmente é chamado de ICAPS[2]. Juntamente com essa competição, foi desenvolvida uma linguagem de definição de domínios com o intuito de tornar mais preciso o processo de comparação entre diferentes planejadores. Tal linguagem, a PDDL, é apresentada na seção 2.3.

2.3 A linguagem PDDL

A linguagem PDDL[3] [27] surgiu em 1998 com o intuito de fornecer um padrão para declaração de problemas e domínios, permitindo uma comparação mais precisa entre diferentes planejadores.

Resultante das linguagens STRIPS e ADL [28], PDDL apresenta características de representação como ações no estilo STRIPS, efeitos condicionais, quantificação universal em universos dinâmicos, axiomas de domínio, ações hierárquicas, e múltiplos problemas em múltiplos domínios. Tais características podem ser brevemente descritas conforme abaixo:

- As ações no estilo STRIPS, conforme vistas na seção 2.1, possuem em sua descrição:

[1] International Conference on Artificial Intelligence Planning Systems
[2] International Conference on Automated Planning and Scheduling
[3] The Planning Domain Definition Language

uma lista (pre) de pré-condições e duas listas (pos e del) contendo os literais que devem ser adicionados e removidos após a execução da respectiva ação;

- Efeito condicional significa que os efeitos de uma determinada ação podem variar de acordo com as características dos objetos recebidos como parâmetros. Pode-se, por exemplo, declarar uma ação que ora possui dois literais como efeito, ora não possui nenhum;

- A quantificação universal permite aplicar uma ação a um conjunto de objetos presentes em um determinado ambiente. Considere, por exemplo, um domínio de transporte de objetos entre diferentes salas e um robô. Neste caso, tal característica possibilita declarar uma ação na qual o robô pegará todos os objetos da cor azul;

- Axiomas de domínio permitem derivar relações entre literais de acordo com a ocorrência de derminadas situações. Em outras palavras, os axiomas permitem deduzir novas informações a partir da descrição do estado vigente;

- Ações hierárquicas é uma funcionalidade que permite tratar problemas no mesmo estilo dos planejadores hierárquicos, como Sipe [29], O-Plan [30] e UMCP [31]. Nesse estilo, o objetivo é conseguir compor ações abstratas a partir de ações primitivas disponíveis;

Uma característica importante da linguagem PDDL é o fato de sua notação ser neutra, o que significa que não são fornecidos "conselhos" para os sistemas planejadores. Os conselhos podem ser vistos como informações adicionais com a finalidade de guiar a busca por uma solução. Por exemplo, poderiam ser disponibilizadas informações do tipo "quais ações usar" para alcançar determinados objetivos, ou então, "quais ações usar" em determinadas situações.

A PDDL permite tanto definir domínios como também instanciar problemas de domínios previamente definidos. Para expressar um determinado domínio, a linguagem PDDL possibilita declarar o nome do mesmo, o subconjunto de características da linguagem exigido, os predicados existentes, e as ações disponíveis. Uma ação é definida por: seu nome, seus parâmetros, suas pré-condições e seus efeitos.

Já a representação de um problema em PDDL é composta por: nome do problema, domínio ao qual o mesmo pertence, conjunto de objetos existentes, um estado inicial e um estado final.

Conforme visto na seção 2.1, um problema de planejamento é definido pela tripla $P =< O, I, G >$. Sendo assim, serão apresentados exemplos utilizando a linguagem PDDL para definir O, I e G.

Um exemplo de domínio é o *gripper*. Tal domínio consiste de um robô, com uma ou mais garras, que deve movimentar objetos entre diferentes salas. Embora o mesmo não explore todas as funcionalidades da linguagem, possui aspectos de declaração muito semelhantes aos domínios os quais serão explorados pelo algoritmo *GAPNet* da seção 4.2.

O arquivo que descreve o domínio gripper é listado abaixo contendo, a cada trecho, uma explicação das informações nele contidas.

```
(define (domain gripper-strips)
   (:requirements :typing)
   (:predicates (at-robby ?r - room)
                (at ?b - obj ?r - room)
                (free ?g - gripper)
                (carry ?o - obj  ?g - gripper))
```

Nesta primeira parte foram definidos: o nome do domínio (*gripper-strips*), as características exigidas da linguagem (*typing*) e quatro predicados: *at-robby*, *at*, *free* e *carry*. Tais predicados podem ser brevemente descritos como:

- *at-robby*: informa que o robô encontra-se em uma determinada sala;
- *at*: informa que um determinado objeto encontra-se em uma determinada sala;
- *free*: informa que uma determinada garra está livre;
- *carry*: informa que um determinado objeto encontra-se em uma determinada garra.

```
(:action move
    :parameters  (?from - room ?to - room)
    :precondition (and  (at-robby ?from) )
    :effect (and  (at-robby ?to)
                  (not (at-robby ?from))))
```

A partir deste trecho é iniciada a declaração das ações disponíveis no domínio. Cada uma delas contém um nome, um conjunto de parâmetros, de pré-condições e de efeitos. Neste caso, a primeira ação declarada possui o nome *move* e é utilizada para mover o robô entre duas salas. Essa ação requer como parâmetro dois objetos (*from* e *to*) do tipo *room* e, para ser executada, exige que o robô esteja presente na sala *from*. Uma vez satisfeita essa condição, a ação é executada e o robô passará a estar na sala *to*. Conforme visto na seção 2.1, os efeitos podem tanto adicionar características como remove-las. No caso da ação *move* o efeito "robô na sala *to*" é adicionado, enquanto o efeito "robô na sala *from*" é removido.

```
(:action pick
    :parameters (?o1 - obj ?r1 - room ?g1 - gripper)
    :precondition (and  (at ?o1 ?r1) (at-robby ?r1)
                        (free ?g1)
                  )
    :effect (and (carry ?o1 ?g1)
                 (not (at ?o1 ?r1))
                 (not (free ?g1))))
```

A segunda ação deste domínio é nomeada *pick* e tem como função pegar um determinado objeto com um garra do robô. A ação *pick* possui três parâmetros, sendo o primeiro deles (*o1*) do tipo *obj*, o segundo (*r1*) do tipo *room* e o terceiro (*g1*) do tipo *gripper*. Suas condições para execução são que tanto o objeto quanto o robô estejam na mesma sala, e

ainda que garra esteja livre. Após executar esta ação a garra passa a não estar mais livre, o objeto deixa de estar na sala e passa a estar na garra do robô.

```
   (:action drop
       :parameters  (?o1 - obj   ?r1 - room ?g1 - gripper)
       :precondition  (and  (carry ?o1 ?g1) (at-robby ?r1))
       :effect (and (at ?o1 ?r1)
                    (free ?g1)
                    (not (carry ?o1 ?g1))))
)
```

A última ação desse domínio é a ação nomeada *drop*, a qual tem por finalidade largar um objeto presente em uma das garras do robô. Possui parâmetros iguais aos da ação *pick* (*o1*, *r1* e *g1*), entretanto, exige que o objeto *o1* esteja na garra *g1* e também que o robô esteja na sala *r1*. A execução dessa ação faz com que o objeto passe a estar na sala e que a garra passe a estar livre.

Tendo como referência o domínio *gripper-strips*, é interessante destacar duas seções no arquivo de descrição de domínios: a primeira delas refere à declaração de predicados e a segunda à declaração de ações. Essas informações são importantes visto que permitem consultar e modificar características em um determinado estado do problema.

O exemplo a seguir descreve um problema para o domínio *grippper-strips* contendo, assim como na descrição anterior, uma breve explicação para cada trecho.

```
(define (problem prob01)
   (:domain gripper-strips)
```

No topo do arquivo de descrição é inserido o nome do problema a ser descrito e, logo após, a primeira seção (*:domain*) define a qual domínio o mesmo pertence.

```
   (:objects r1 r2 - room
             b1 b2 b3 b4 - obj
             gleft gright - gripper )
```

A seção *:objects* define quais objetos existem no problema, utilizando uma versão "tipada" conforme declarado (*:requirements :typing*) na descrição do domínio. Essa característica vincula um tipo (característica) a cada objeto criado, permitindo que ações sejam instanciadas apenas com parâmetros adequados. Neste caso, percebe-se a declaração de dois objetos (*r1* e *r2*) do tipo *room*, quatro objetos (*b1*, *b2*, *b3* e *b4*) do tipo *obj*, e dois objetos (*gleft* e *gright*) do tipo *gripper*.

```
(:init
        (free gleft)
        (free gright)
        (at-robby r1)
        (at b1 r1)
        (at b2 r1)
        (at b3 r1)
        (at b4 r1))
```

A seção *:init* permite declarar quais características estarão presentes no estado inicial do problema, utilizando para isso os predicados definidos no arquivo de descrição do domínio. Conforme este exemplo, o estado inicial é definido por: as garras *gleft* e *gright* estão livres e, tanto o robô quanto as bolas (*b1*, *b2*, *b3* e *b4*), estão na sala *r1*.

```
(:goal (and (at b4 r2)
            (at b3 r2)
            (at b2 r2)
            (at b1 r2)))
)
```

A última seção (*:goal*) diz respeito à declaração do estado objetivo do problema. Assim como na seção anterior, utiliza predicados para definir as características desejadas. Neste caso o objetivo a ser alcançado é que todas as bolas (*b1*, *b2*, *b3* e *b4*) estejam presentes na sala *r2*. Note que nada é informado sobre a localização do robô, demonstrando que este

estado não necessariamente precisa conter uma descrição de todos os objetos existentes no problema. O arquivo de descrição do problema termina, assim, no final da seção *:goal*.

Essas duas exemplificações mostram como utilizar a notação da linguagem PDDL com o intuito de descrever um domínio e também um problema de planejamento clássico. Esse conjunto de informações deve ser salvo em arquivos os quais servirão de entrada para os sistemas planejadores. Para isso, normalmente é criado um arquivo com o nome do domínio e outro com o nome do problema, ambos contendo a extensão "pddl".

Seguindo a idéia de qualificar a comparação entre planejadores e, ao mesmo tempo, disponibilizar um ambiente para desenvolvimento dos mesmos, a seção 2.4 apresenta o IPE.

2.4 IPE - Ambiente de Planejamento Ipê

O Ambiente de Planejamento Ipê (IPê Planning Environment) foi desenvolvido por Marynowski [32] no LIC (laboratório de inteligência computacional do Departamento de Informática da Universidade Federal do Paraná). O ambiente surgiu com o intuito de fornecer tanto uma plataforma padrão para o desenvolvimento de sistemas planejadores, como também um meio mais preciso para a comparação de desempenhos entre os mesmos.

Embora os planejadores utilizem a linguagem PDDL como referência, existem outros aspectos que podem ser padronizados a fim de tornar mais preciso o processo de comparação. Pode-se citar:

- O procedimento responsável por realizar uma análise sintática dos arquivos de entrada e conseqüente importação das informações para estruturas internas pode ser o mesmo para diferentes planejadores;

- Planejadores que se baseiam em mesmas representações podem compartilhar códigos ao gerar a representação necessária. Por exemplo: dado que existem diversos planejadores que têm como base a estrutura do grafo de planos, não é necessário que cada planejador reescreva um método para geração dessa estrutura.

Na verdade, o ambiente IPE oferece uma coleção de classes que permite o desenvolvimento de um planejador sem a necessidade de reescrever código para procedimentos já existentes. A seguir serão descritas algumas classes disponíveis:

- Analisador Sintático: faz análise dos arquivos de domínio e problema descritos na linguagem PDDL, disponibilizando acesso a estas informações;
- Problema: instancia ações e predicados de um determinado problema;
- Grafo de Planos: cria a estrutura do grafo de planos (inclusive o relaxado);
- Petri Net: cria uma representação baseada em redes de Petri;
- Formula Sat: cria uma representação baseada em fórmulas SAT;

Agora, com base nas classes disponibilizadas pelo ambiente IPE, imagine que se queira criar um planejador baseado no grafo de planos. O que será realmente necessário desenvolver será apenas o algoritmo o qual buscará por uma solução no grafo de planos. A vantagem é que, a partir do momento em que existem vários planejadores baseados em uma mesma representação, todos poderão ser comparados de forma mais precisa dado o fato de utilizarem uma mesma base de desenvolvimento. A medida em que os planejadores são desenvolvidos, os mesmos são inseridos no ambiente IPE como sendo uma nova classe.

Sendo assim, existem classes que são o sistema planejador propriamente dito:

- Graphplan: faz uma busca exaustiva no grafo de planos;
- AGPlan: utiliza algoritmos genéticos para resolver um problema de planejamento, tendo como referência o grafo de planos;
- Petriplan: aplica programação inteira na solução de um problema de alcançabilidade em redes de Petri;
- FFPlan: é uma implementação simplificada do *FF*, sem considerar a busca local.

O *PetriPlan* [14][15], por exemplo, é um planejador que gera uma rede de Petri baseada nas mesmas idéias do grafo de planos gerado pelo *Graphplan*. A partir do momento em que se tem uma rede de Petri a qual modela um problema de planejamento, este mesmo pode ser tratado como um problema de alcançabilidade em redes de Petri (ver seção 2.5.1).

Desde o surgimento do *Petriplan*, os pesquisadores do LIC têm focalizado seu trabalho no desenvolvimento de novos planejadores baseados em redes de Petri. Visto que o algoritmo proposto no capítulo 4 também tem como base essa estrutura, a mesma será utilizada como exemplo em uma breve descrição de como desenvolver um planejador no ambiente IPE.

O primeiro procedimento a ser executado por um sistema planejador é realizar uma análise sintática dos arquivos de entrada, os quais contém as descrições de domínio e problema a serem tratados. Para isso, deve-se instanciar um objeto da classe *parser*, informando os nomes dos arquivos de domínio e de problema.

Uma vez obtido sucesso na fase de análise sintática, deve-se passar para etapa de instanciação de ações e predicados referentes ao problema em questão. Este processo é feito pela criação de um objeto da classe *problem*, a qual necessita como parâmetro o objeto da classe *parser* gerado na fase anterior.

A próxima etapa envolve uma decisão muito importante com relação ao sistema planejador, que é a decisão sobre qual das representações disponíveis utilizar. Neste caso, como optou-se pelas redes de Petri, a classe a ser usada é a *petrinet*. Para que essa classe possa gerar a estrutura necessária, é preciso que seja passado como parâmetro o objeto da classe *problem*.

A geração de uma representação, a qual modela um problema de planejamento, permite ao desenvolvedor abstrair determinadas peculiaridades do problema e focar-se unicamente na estrutura a qual deve ser tratada. Na modelagem com redes de Petri, por exemplo, isso significa que o projetista do sistema planejador deverá preocupar-se em resolver um problema de alcançabilidade na rede obtida.

Tendo isso em mente, deve-se desenvolver uma classe que é dita ser um resolvedor. Um resolvedor pode ser visto como o núcleo de um sistema planejador, o qual recebe como

parâmetro uma representação e devolve, caso encontre, um plano. Seguindo o exemplo no ambiente IPE, o último passo a ser realizado é instanciar um objeto resolvedor a partir do objeto obtido pela classe *petrinet*.

Essa descrição de uso do ambiente IPE, mostra como o mesmo possui vantagens tanto no sentido de comparação de desempenho quanto no sentido de disponibilizar uma plataforma de fácil desenvolvimento. A existência de uma coleção de classes a serem compartilhas não somente incentiva o desenvolvimento, como também aumenta a confiabilidade e a organização dos novos sistemas gerados.

Como será visto no capítulo 4, o algoritmo a ser proposto completa a última fase de desenvolvimento descrita por esta seção. Visto que o mesmo pertence ao ambiente IPE e utiliza uma modelagem com redes de Petri, a próxima seção será dedicada à definição conceitual dessas redes e também à análise das características da classe *petrinet*.

2.5 Representação com redes de Petri

Para compreender o desenvolvimento de um planejador baseado na representação de redes de Petri, é preciso conhecer tanto o formalismo de redes de Petri quanto a classe *petrinet* do ambiente IPE. As seções 2.5.1 e 2.5.2 suprem, respectivamente, essas necessidades.

2.5.1 Redes de Petri

As redes de Petri(RdP) são um formalismo matemático proposto por Carl Adam Petri [33] em 1962 e são muito utilizadas para modelar sistemas a eventos discretos, permitindo tratar características como concorrência e paralelismo. Uma RdP é um grafo dirigido composto por dois elementos principais: transição e lugar. Enquanto uma transição representa a ocorrência de um evento no sistema, um lugar representa uma característica, como por exemplo, status, recurso, produto, informação, etc.

Formalmente, uma RdP é definida pela quintupla $N = (P, T, Pre, Pos, M_0)$ onde:

- $P = \{p_1, p_2, \ldots, p_n\}$ é uma lista finita de lugares;
- $T = \{t_1, t_2, \ldots, t_m\}$ é uma lista finita de transições;

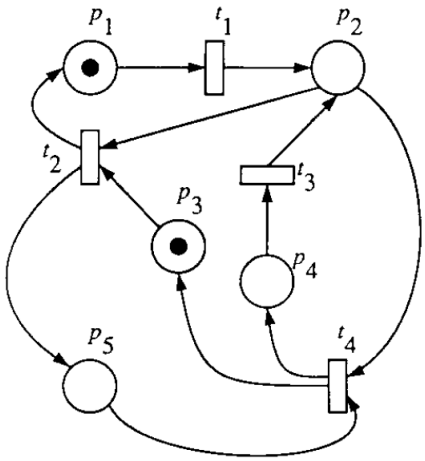

Figura 2.2: Rede de Petri - Exemplo

- Pre é uma matriz de incidência anterior: $P \times T \rightarrow \mathbb{N}$ (valor do arco $p \rightarrow t$);
- Pos é uma matriz de incidência posterior: $P \times T \rightarrow \mathbb{N}$ (valor do arco $t \rightarrow p$);
- $M_0 : P \rightarrow \mathbb{N}$ é a marcação inicial da rede.

O funcionamento de uma rede de Petri se dá pelo disparo de transições e conseqüente alteração da marcação da rede, ou seja, os estados de um sistema são alterados de acordo com a ocorrência de eventos no mesmo. A figura 2.2 ilustra uma rede de Petri com 5 lugares, 4 transições, e marcação inicial $M_0 = \{1, 0, 1, 0, 0\}$.

Pode-se notar que os arcos da figura 2.2 não apresentam, visualmente, um valor explícito. Esta é uma convenção usada para representar arcos com valores iguais a 1 (um). Quando todos os arcos de uma rede de Petri possuem valor 1, a mesma é dita ser uma rede ordinária. A existência de valores iguais a zero nas matrizes de incidência anterior e posterior denotam a ausência de arcos.

Diz-se que uma transição t está habilitada a disparar se $\forall p \in P$, $M(p) >= Pre(p, t)$. Sendo assim, segundo a rede marcada da figura 2.2, percebe-se que somente a transição t_1 pode ser disparada.

Após o disparo de uma transição t, ocorrerá um consumo de $Pre(p, t)$ marcas e uma produção de $Pos(p, t)$ marcas para todo $p \in P$. Por exemplo, o disparo de t_1 na rede da figura 2.2, modificaria a marcação da mesma para $M' = \{0, 1, 1, 0, 0\}$ e conseqüentemente habilitaria t_2.

Sendo assim, a nova marcação para cada lugar em uma rede a partir do disparo de uma transição t pode ser expressa como: $\forall p \in P, M'(p) = M(p) - Pre(p,t) + Pos(p,t)$.

Considerando P de tamanho n e T de tamanho m, obtemos da relação $P \times T$ uma matriz de dimensão $n \times m$. A notação $Pre(.,t)$ faz referência a coluna t da matriz Pre, representando os arcos de entrada da transição t (inclusive os de valor igual a zero). Da mesma forma, a notação $Pos(.,t)$ faz referência a coluna t da matriz Pos, representando os arcos de saída da transição t (inclusive os de valor igual a zero).

A partir dessa nova notação, pode-se dizer que uma transição t está habilitada se $M >= Pre(.,t)$, já que tanto M quanto $Pre(.,t)$ são vetores coluna de dimensão igual a n. Agora, a nova marcação obtida com o disparo de t, pode ser definida como
$M' = M + Pos(.,t) - Pre(.,t)$.

Visando disparar uma seqüência de transições, considere:

- $C = Pos - Pre$, chamada de matriz de incidência;
- $\overline{s} : T \rightarrow \mathbb{N}$, chamado vetor característico.

O vetor característico $\overline{s}$ informa quantas vezes cada transição será disparada em uma seqüência. Por exemplo, se $\overline{s}(1) = 0$ e $\overline{s}(2) = 3$, significa que a transição t_1 não será disparada e que a transição t_2 será disparada três vezes.

A equação obtida para determinar a nova marcação M_n de uma rede de Petri, dado o disparo de uma seqüência de transições, é conhecida como *equação fundamental* de N:

$$M_n = M + C.\overline{s} \tag{2.2}$$

Considere a seguinte questão: dado um sistema modelado com uma rede de Petri, quais são os estados alcançáveis pelo mesmo a partir de uma marcação inicial M_0? Essa pergunta está relacionada a um problema de alcançabilidade.

Um problema de alcançabilidade em Redes de Petri é assim definido: dada uma RdP R com uma marcação inicial M_0, existe uma seqüência de disparos s que altera a marcação da rede para M'? Formalmente, existe s que satisfaça $M_0 \xrightarrow{s} M'$ para uma rede R ?

A classe *petrinet*, a qual será vista a seguir, gera uma rede de Petri para um problema de planejamento e possibilita que esse seja tratado como sendo um problema de alcançabilidade em redes de Petri.

2.5.2 A classe *petrinet*

Em 2000, Silva [15] propôs um planejador chamado *petriplan* o qual resolve um problema de alcançabilidade em Redes de Petri utilizando programação inteira. Como a rede de Petri considerada modela um problema de planejamento em inteligência artificial, a solução para o problema de alcançabilidade representa uma solução para o problema de planejamento em questão. Para tornar isso viável foi implementado um procedimento que faz a tradução de um grafo de planos para uma rede de Petri.

Posteriormente, com o surgimento de novas versões do *petriplan*, foi desenvolvida uma classe chamada *petrinet* a qual constrói uma rede de Petri diretamente a partir dos arquivos de descrição em PDDL, sem a necessidade de passar pela representação do grafo de planos.

As características da rede gerada são muito parecidas com as de um grafo de planos, porém, o formalismo matemático no qual as redes de Petri se baseiam passam a ser um grande diferencial na hora de analisar a dinâmica de um problema de planejamento. Conforme visto na seção 2.5.1, a transição entre estados (marcações) de um sistema pode ser facilmente executada a partir de cálculos matriciais, conforme a equação 2.2.

A figura 2.3 mostra uma rede de Petri gerada pela classe *petrinet*. Tal estrutura é resultado de uma modelagem de um problema pertencente ao domínio *mundo dos blocos*[4], cujo objetivo é desempilhar sobre a mesa uma pilha formada por três blocos. Como pode ser observado, essa rede apresenta 10 camadas alternadas entre camadas de lugares e camadas de transições. Considerando da esquerda para a direita temos as camadas: $L_0, L_1, \ldots, L_9$.

Como as características da rede se repetem a cada 4 camadas, será feita uma análise levando-se em conta somente as 4 primeiras (L_0, L_1, L_2, L_3). L_0 representa as proposições

[4]Uma descrição completa do domínio *mundo dos blocos* é apresentada na seção 4.3

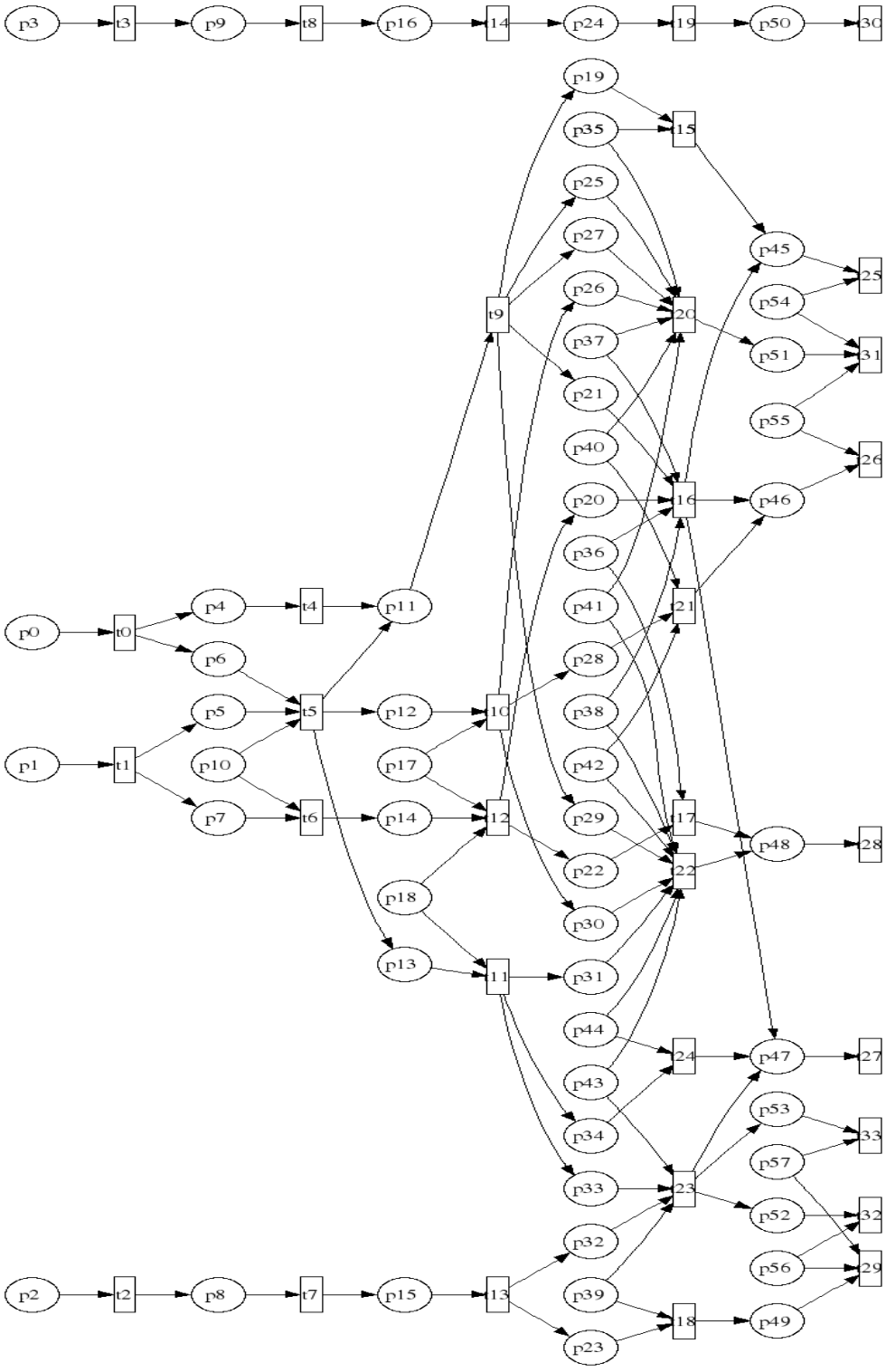

Figura 2.3: Rede de Petri gerada pela classe *petrinet*

existentes em um estado, enquanto que a L_2 é constituída por várias instâncias das mesmas proposições de L_0. Por exemplo: imagine que em L_0 exista p, e que existam três ações que necessitam de p para serem executadas. Sendo assim, em L_2 haverá três réplicas de p.

L_1 é responsável por fazer as cópias necessárias de cada proposição existente em L_0, gerando-as em L_2. Já L_3 é a camada das ações disponíveis para um determinado estado. É interessante observar que o número de camadas da rede gerada não é múltiplo de 4, visto que não existem, no final da rede, as correspondentes à categoria de L_2 nem de L_3.

De acordo com o que foi apresentado na seção 2.2, a dificuldade para se encontrar um plano é a existência de conflitos entre ações de uma mesma camada. Enquanto no grafo de planos os mesmos são representados por ligações entre as respectivas ações conflitantes (metainformação), já em uma rede de petri gerada pela classe *petrinet*, um conflito faz parte da estrutura da rede e é representado por um lugar p com as seguintes características:

- $M(p) = 1$;
- $\forall t \in T, Pre(p,t) = 0$
- Existem exatamente duas transições $t \in T$ onde $Pos(p,t) = 1$. Para todas as outras transições $t \in T$, $Pos(p,t) = 0$.

Ou seja, um conflito é representado por um lugar com marcação igual a um, contendo zero arcos de entrada e exatamente dois arcos de saída. Tais arcos estão ligados a duas transições as quais passam a ser mutuamente exclusivas, característica identificada como *mutex*.

Nessa rede há dois tipos de *mutex*: *mutex* de proposição (literal) e *mutex* de ação. Os de proposição estão presentes nas camadas de mesma categoria que L_0, enquanto que os de ação estão presentes nas camadas de mesma categoria que L_2. Uma vez considerados os *mutex* de ação, não é necessário preocupar-se com os de proposição.

Conhecidas as características da rede de Petri gerada pela classe *petrinet*, é interessante perceber onde está a vantagem de se obter uma estrutura como esta. Suponha que uma nova versão da classe *petrinet* seja adaptada para modelar problemas de planejamento não

clássico, incorporando, por exemplo, o uso de recursos e também características temporais. Nesse momento, um sistema planejador o qual resolve apenas problemas clássicos poderá ser facilmente reutilizado ou adaptado para solucionar problemas não clássicos, visto que continuará tendo que tratar um problema de alcançabilidade em redes de Petri.

Além dessa característica, o fato de poder tratar um problema de planejamento como sendo um problema de alcançabilidade em redes de Petri já traz benefícios para o desenvolvedor, permitindo-lhe abstrair peculiaridades do problema representado e focalizar-se simplesmente na estrutura a qual está sendo analisada. O capitulo 4, por exemplo, propõe um sistema planejador o qual permite resolver problemas de alcançabilidade nas redes de Petri geradas pela classe *petrinet*. Tendo isso em vista, o capítulo 3 descreve o paradigma dos algoritmos genéticos, necessário para compreender o planejador proposto.

CAPÍTULO 3

COMPUTAÇÃO EVOLUTIVA

Com o intuito de completar a base teórica necessária para a compreensão do sistema planejador proposto pelo capítulo 4, este capítulo é reservado para discutir os conceitos sobre computação evolutiva. A computação evolutiva tem sua base na capacidade de adaptação dos seres vivos, buscando importar tal característica para os sistemas de computação e usá-la como princípio na solução de diversos problemas. Nessa área encontram-se vários paradigmas, como por exemplo, programação genética, programação evolutiva e algoritmos genéticos. Neste trabalho é abordado o último.

3.1 Algoritmos genéticos

O paradigma dos algoritmos genéticos foi proposto por John Holland durante os anos 60, a partir de um trabalho em conjunto com seus alunos da Universidade de Michigan. Em 1975 Holland publicou o livro *Adaptation in Natural and Artificial Systems* [18], tendo como referência a teoria da seleção natural de Charles Darwin [34], a qual foi utilizada para justificar a evolução e a adaptação das espécies ao longo do tempo.

A teoria da seleção natural diz que:

> *Os indivíduos mais adaptados ao meio em que se encontram, tendem a se reproduzir mais do que os restantes.*

O que acontece na natureza, e foi percebido por Charles Darwin, é que as características de uma população de certa espécie mudam de geração para geração, adaptando-se às condições ambientes. Um dos exemplos por ele citado foi o de uma população de girafas que se adaptou ao ambiente em virtude do tipo de alimentação disponível. Em uma determinada época existiam tanto girafas de pescoço longo quanto girafas de pescoço curto, porém, com o passar do tempo e com a diminuição de vegetação rasteira, aquelas

que tinham pescoço curto começaram a ser minoria no grupo.

Tal acontecimento indicava que a necessidade de alimentação fez com que as características que determinavam a existência de pescoços mais longos fossem predominando sobre as demais características. Pôde-se notar também que, eventualmente, ocorriam mutações genéticas (pequenas variações no material genético de um ser vivo) nas populações. Embora nem sempre tais mudanças implicassem boas características, as mesmas garantiam que certa diversidade sempre existisse em uma determinada população.

A natureza mostrava-se, então, detentora de uma magnífica técnica de adaptação para os seres vivos. Logo, por que não inspirar-se nela para resolvermos problemas de nossa realidade? Essa é a proposta dos algoritmos genéticos.

Neste paradigma encara-se o problema que está sendo tratado como sendo um ambiente no qual vivem determinados organismos vivos. Cada organismo vivo (indivíduo) representa uma possível solução para o problema. Neste caso, dizer o quão adaptado um indivíduo é com relação ao ambiente é o mesmo que dizer o quão bem uma solução resolve o problema em questão.

Tendo isso em vista, é necessário que uma solução seja codificada em forma de um cromossomo (seqüência de genes). Geralmente, a representação utilizada é a binária, ou seja, uma seqüência de genes cujos valores podem ser zero ou um. Outro ponto muito importante é que, a partir de uma seqüência de genes, deve ser possível a aplicação de um procedimento o qual efetua o cálculo de qualidade desta solução.

Esse cálculo de qualidade, ou grau de adaptação de um indivíduo (solução) em relação ao ambiente (problema), é conhecido como função de aptidão. A função de aptidão é particular de cada problema e é responsável por guiar a evolução de uma determinada população.

Até aqui foi visto que um problema precisa ser adaptado para ser um ambiente e que uma solução precisa ser codificada como sendo um indivíduo (cromossomo) o qual viverá nesse ambiente. Agora, buscando simular o processo de evolução natural, é descrito um conjunto de operadores genéticos responsáveis por combinar e variar as características (genes) da população ao longo de diversas gerações. Tais operadores são conhecidos como

operador de cruzamento e operador de *mutação*, os quais serão vistos na seção 3.1.3 e na seção 3.1.4 respectivamente.

Antes de apresentar os operadores genéticos e demais métodos necessários durante o processo de evolução, serão definidos alguns parâmetros de entrada e também um pseudocódigo o qual representa a execução de um algoritmo genético.

Os parâmetros podem variar um pouco de acordo com cada implementação, porém, os principais são:

- Tamanho da população: número de indivíduos, isto é, soluções candidatas;
- Taxa de cruzamento: percentual de indivíduos os quais serão submetidos ao operador de cruzamento a cada geração;
- Taxa de mutação: percentual de indivíduos os quais serão submetidos ao operador de *mutação* a cada geração;
- Taxa de elitismo: percentual de indivíduos os quais têm lugar garantido na próxima geração;
- Tipo de seleção: define o método o qual seleciona indivíduos para a aplicação do operador de cruzamento;
- Tipo de cruzamento: define qual será o tipo de cruzamento utilizado;
- Número de gerações: uma das condições de parada de um algoritmo genético.

Um pseudocódigo do funcionamento de um algoritmo genético é descrito pelo algoritmo 3.

Geralmente a população inicial é gerada aleatoriamente, entretanto, para alguns problemas, pode-se conduzir a formação das soluções candidatas a partir de uma característica desejável para seu genótipo. Já o critério de parada está associado ao número máximo de gerações a serem executadas ou a um valor aceitável para a avaliação de *aptidão*.

Algoritmo 3 Pseudocódigo de um algoritmo genético

```
1: Gere uma população inicial P;
2: Avalie a população P;
3: while (não satisfizer critério de parada) do
3:     Execute o método de seleção e aplique o operador de cruzamento nos indivíduos
       selecionados, salvando-os em P';
3:     Aplique o operador de mutação na população P, salvando o resultado em P';
3:     Avalie os indivíduos de P';
3:     Atualize a população P a partir de P e P';
4: melhor indivíduo;
```

As próximas seções completam o conceito dos algoritmos genéticos, descrevendo a função de aptidão (3.1.1), seleção natural (3.1.2), operadores genéticos (3.1.3 e 3.1.4) e atualização da população (3.1.5).

3.1.1 Função de aptidão

Assim como já comentado anteriormente, a função de aptidão é particular de cada problema tratado. Genericamente, o que se pode dizer é que esta função é a qualidade de um indivíduo com relação ao meio em que o mesmo se encontra, ou seja, o seu grau de adaptação às condições ambientes. Em outras palavras, representa o quão aceitável é uma solução para o problema que está sendo tratado.

3.1.2 Seleção natural

O método de seleção é responsável por eleger indivíduos os quais serão submetidos ao operador de cruzamento. Durante esse processo deve-se levar em conta a teoria da seleção natural, fazendo com que indivíduos melhores adaptados tenham maiores chances de serem selecionados.

Existem diversas variações de como selecionar esses indivíduos. Nesta seção serão apresentadas duas delas: seleção por roleta e seleção por torneio.

O método da roleta estipula um intervalo no qual estarão distribuídos todos os indivíduos de uma população. Cada indivíduo receberá uma porção desse intervalo de

acordo com a sua aptidão, seguindo a seguinte equação:

$$I_i = F(i) / \sum_{f=1}^{n} F(f) \tag{3.1}$$

Onde I_i é a porção do intervalo recebida pelo indivíduo i, $F(i)$ é a aptidão do indivíduo i, e n é o tamanho da população. Dessa forma, para selecionar um indivíduo, basta gerar um número aleatório entre 0 e $\sum_{f=1}^{n} F(f)$. A probabilidade de um indivíduo ser selecionado é diretamente proporcional a sua avaliação de aptidão.

Já a seleção por torneio necessita de um parâmetro adicional: o tamanho do torneio. Geralmente esse parâmetro é um percentual do tamanho da população, assim como as taxas de cruzamento e de *mutação*, e determina quantos indivíduos participarão de um torneio. Para selecionar k indivíduos é necessário executar k torneios de tamanho t cada um, conforme mostra o algoritmo 4.

Algoritmo 4 Seleção por torneio

```
1: Crie uma lista L, vazia;
2: for S = 1 to k do
3:     Selecione aleatoriamente t indivíduos;
4:     Adicione o indivíduo de melhor aptidão a lista L;
5: return L;
```

Após executar o método de seleção, independente de qual for, deve-se organizar os indivíduos selecionados em pares e submetê-los ao operador de cruzamento.

3.1.3 Operador de cruzamento

O operador de cruzamento é responsável por combinar o material genético de dois indivíduos (pais), gerando dois descendentes com características de ambos progenitores. Para realizar essa combinação gera-se um ponto de corte aleatório no cromossomo, determinando quais genes formarão os novos indivíduos. O cruzamento pode ser classificado de acordo com o número de pontos de cortes exigidos para a combinação, conforme ilustram as figuras 3.1 e 3.2

Existe também uma variação do cruzamento clássico, o qual utiliza uma máscara de

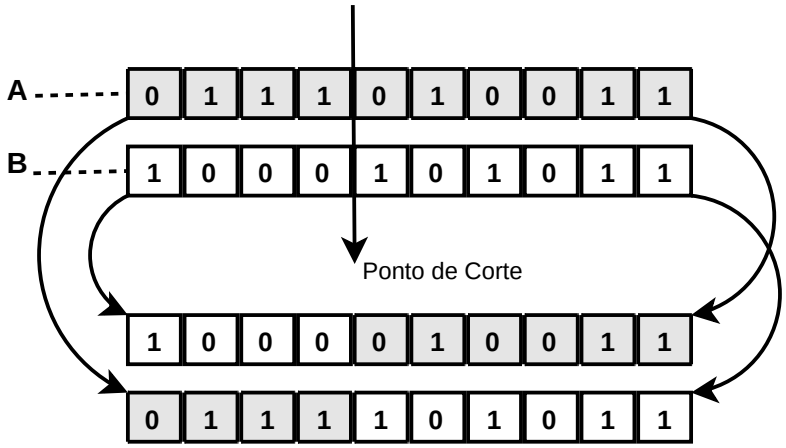

Figura 3.1: Operador de cruzamento, um ponto de corte

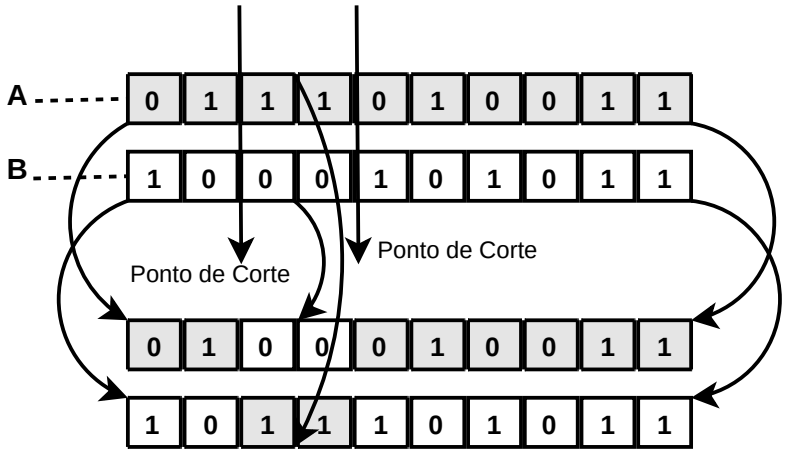

Figura 3.2: Operador de cruzamento, dois pontos de corte

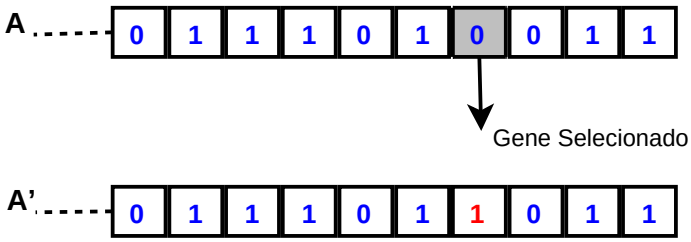

Figura 3.3: Operador de *mutação*

bits para determinar quais genes serão selecionados. Neste caso, apenas os genes marcados como 1 (um) na máscara de bits serão trocados pelo procedimento de cruzamento.

Do ponto de vista de busca em um espaço de soluções, o cruzamento é responsável por percorrer regiões bem amplas e que podem, em muitos casos, compartilhar determinadas características. O número de indivíduos os quais serão submetidos a este operador é definido pela taxa de cruzamento.

3.1.4 Operador de *mutação*

O operador de *mutação* é responsável por garantir diversidade na população e também por realizar pequenos ajustes nas soluções exploradas. Na representação binária, por exemplo, esse operador efetua uma inversão de um determinado bit (gene). Já para o caso de representações diferentes, deve-se definir alguma regra para modificação de um gene. A figura 3.3 ilustra a ocorrência de uma mutação no cromossomo.

Ao contrário do operador de cruzamento, o operador de *mutação* não necessita de uma prévia seleção de indivíduos. Neste caso, o próprio método é responsável por selecionar aleatoriamente (de acordo com a *taxa de mutação*) tantos genes quantos forem necessários para realizar a operação.

3.1.5 Atualização da população

Tradicionalmente a população deve permanecer com o mesmo tamanho durante todas as gerações do processo evolutivo, porém, após a aplicação dos operadores genéticos o número de indivíduos existentes é maior do que o realmente necessário. A forma de decidir quais indivíduos devem permanecer pode variar de acordo com cada implementação e também

de acordo com os parâmetros utilizados.

No caso de utilizar o parâmetro relacionado ao elitismo, a nova população pode ser formada pela elite da geração anterior e os melhores indivíduos gerados pelos operadores genéticos. Outra opção, por exemplo, é formar parte da nova população com os melhores indivíduos disponíveis e outra parte com alguns indivíduos cuja aptidão não seja tão expressiva. Essa segunda opção, opta por indivíduos medianos visando manter certa diversidade na população e evitar uma convergência prematura da mesma.

Independente da forma adotada, deve-se observar como a população está evoluindo ao longo das gerações. O ideal é que, no início, a população apresente indivíduos bem diversificados a fim de permitir uma boa análise de todo o espaço de busca. Aos poucos, a população deve começar a convergir em torno da solução desejada.

Este capítulo, o qual apresentou o conceito dos algoritmos genéticos, forma a base necessária para compreender o sistema planejador *GAPNet*, descrito no capítulo 4.

CAPÍTULO 4

GAPNET

Neste capítulo será apresentado o GAPNet (Genetic Algorithm for Reachability on Petri Nets), o qual propõe uma solução para o problema de planejamento em inteligência artificial utilizando o paradigma dos algoritmos genéticos. Partindo de um problema modelado com a rede de Petri vista na seção 2.5.2, o GAPNet aplica uma técnica evolucionária para tratar os conflitos existentes na mesma. Tal tratamento permite resolver um problema de alcançabilidade na rede analisada e, conseqüentemente, resolver o problema de planejamento por ela representado.

Enquanto a seção 4.1 analisa alguns planejadores baseados no paradigma dos algoritmos genéticos, a seção 4.2 apresenta o algoritmo GAPNet.

4.1 Planejadores baseados em algoritmos genéticos

Esta seção apresenta alguns planejadores baseados no paradigma dos algoritmos genéticos, fazendo uma análise de seus principais aspectos e resultados obtidos. O algoritmo da seção 4.1.1, desenvolvido por Lecheta [20], mostra como um problema de planejamento pode ser tratado a partir de uma representação STRIPS. Posteriormente, a seção 4.1.2 expõe um algoritmo desenvolvido por Castilho e colegas [21] o qual utiliza a representação do grafo de planos como referência para gerar seus indivíduos (população inicial). Por último, a seção 4.1.3 apresenta o GASAT [23][24], um resolvedor de fórmulas SAT baseado em algoritmos genéticos.

4.1.1 *AgPlan* com STRIPS

O primeiro sistema a ser analisado é o de Lecheta [20], proposto em 2004, e que tem como ponto de partida um problema definido de acordo com a representação STRIPS.

Uma das características do sistema planejador apresentado é que o mesmo não gera

uma árvore de estados para realizar a busca por soluções. Na verdade, ele utiliza todas as ações definidas no domínio analisado como base para geração de seus indivíduos (população inicial). Neste caso, cada gene do cromossomo é representado por uma das ações disponíveis. Ao aplicar um procedimento como esse, significa que não há informações quanto ao número de ações necessárias para formar um plano.

Essa limitação poderia ser contornada de duas formas. A primeira opção era adaptar o algoritmo genético para lidar com cromossomos de tamanhos variáveis, passando para o processo evolutivo a responsabilidade de encontrar o número adequado de ações necessárias. A segunda alternativa, a qual foi utilizada, era estipular uma cota superior para o número de ações e criar uma nova ação denominada NOP (No OPeration). A ação NOP é uma ação que não tem pré-condições nem efeitos, uma ação nula que pode ser executada em qualquer estado do problema.

A partir do uso da cota superior, todos os cromossomos passam a ter o mesmo número de genes. Por outro lado, o número de ações utilizadas em cada cromossomo pode variar de acordo com o número de ações NOP's presentes no mesmo. Isso permite ao algoritmo genético lidar com estruturas cromossômicas de tamanho fixo ao mesmo tempo em que essas estruturas representam, logicamente, planos com tamanhos variados.

Vale ressaltar que essa alternativa não exclui totalmente os problemas. O uso de uma cota superior muito baixa, por exemplo, pode impedir que o algoritmo evolua adequadamente, enquanto que uma cota superior muito alta determinará um espaço de busca muito maior do que o realmente necessário.

Uma vez apresentadas as primeiras decisões tomadas com relação ao planejador, pode-se dar prosseguimento e descrever suas demais características. Pode-se citar, por exemplo, aquelas ligadas à questões como operadores genéticos, métodos de seleção, atualização da população e função de avaliação.

Os operadores genéticos adotados não variam muito dos padrões apresentados na seção 3.1.3 e 3.1.4. Pode-se destacar uma pequena diferença com relação ao operador de mutação. Na seção 3.1.4 foi visto que a mutação geralmente é caracterizada pela inversão de um bit (gene), dado o uso de uma representação binária nos cromossomos.

Entretanto, neste planejador, cada gene pode assumir o valor de qualquer ação disponível no domínio e adicionalmente a ação NOP. Isso determina que a mutação deve consistir de uma troca de uma ação por outra, onde esta nova ação deve ser selecionada aleatoriamente de um conjunto de ações disponíveis.

Em um algoritmo genético a mutação tem a finalidade de garantir diversidade na população e também refinar a busca por determinadas soluções. Percebe-se, no entanto, que neste caso a ocorrência de uma mutação pode implicar variações mais significativas nas características dos indivíduos.

O método de seleção utilizado neste trabalho foi o da roleta, conforme apresentado na seção 3.1.2. Quanto a atualização da população, foi escolhida uma opção a qual substitui 90% dos indivíduos de uma geração para outra. Isso, por conseqüência, significa o uso de uma taxa de elitismo igual a 10%. Ainda com relação a população, foi decidido permitir a existência de indivíduos duplicados na mesma. Essa decisão, segundo o autor, não traria perda de diversidade visto o enorme espaço de busca considerado em suas análises. Entretanto, também foram realizados testes nos quais não se utilizava repetição de indivíduos nem o elitismo.

A função de avaliação adotada procura extrair diversas características a partir da seqüência de genes avaliada. Com essa finalidade, são consideradas questões como: posição da primeira e da última ação executável, número de ações executáveis, número de ações consecutivas executáveis, número de objetivos alcançados e vetor de executabilidade. O vetor de executabilidade é de natureza binária e determina, a partir do cromossomo, uma seqüência de ações aplicável a partir do estado inicial.

Com base nestas informações, a função de avaliação calcula seu valor de retorno a partir da combinação ponderada das seguintes relações:

- objetivos alcançados e o total de objetivos;
- ações executáveis e ações válidas. Uma ação é dita ser inválida quando instanciada com parâmetros de tipos inadequados;
- blocos de ações executáveis e ações válidas.

Embora bem refinada, essa função de aptidão tem dificuldades para tratar ações que não foram consideradas ao formar um plano. Esse detalhe faz com que, muitas vezes, o operador de cruzamento combine material genético não analisado corretamente. Isso, somado ao vasto espaço de busca considerado, resultou em um planejador com tempos de resposta bem elevados.

A justificativa para o baixo desempenho foi a utilização de bibliotecas públicas tanto de planejamento quanto de algoritmos genéticos. A de planejamento, mais especificamente, realiza a leitura dos arquivos de descrição de domínio e problema considerando uma versão não tipada. Isso implica na instanciação de várias ações inválidas, aumentando significativamente o espaço de busca a ser analisado em alguns domínios. Já a biblioteca de algoritmos genéticos é pouco flexível em alguns aspectos, como por exemplo, inserir código intermediário no processo evolutivo.

Embora tais bibliotecas tenham sido reimplementadas com o intuito de melhorar o desempenho apresentado pela abordagem inicial, os resultados obtidos não apresentaram melhoras significativas.

4.1.2 *AgPlan* com grafo de planos

O *agplan*, desenvolvido por Castilho e colegas [21], é um planejador baseado em algoritmos genéticos o qual utiliza a representação do grafo de planos como referência para geração de seus indivíduos (população inicial). Essa proposta complementa o trabalho de Lecheta, fornecendo uma codificação de maior qualidade para os cromossomos e limitando o espaço de busca a uma região bem menor.

Como foi visto na seção anterior, optar por gerar soluções candidatas diretamente a partir da representação STRIPS não torna eficiente o processo de busca por uma solução. Uma vez que o número de ações instanciadas é muito maior do que o necessário, o processo evolutivo despende muito tempo de processamento analisando regiões dos espaço de busca as quais não precisariam ser consideradas.

O grande diferencial do algoritmo proposto por Castilho e colegas é a forma com que as ações são selecionadas para gerar um indivíduo. Para isso, são levadas em consideração

as camadas de ações presentes na versão relaxada do grafo de planos, onde cada camada produzirá uma região diferente no cromossomo gerado.

Dessa forma, o número de regiões de um cromossomo será sempre igual ao número de camadas de ações existentes no grafo de planos analisado. É importante ressaltar que estas regiões são identificadas apenas no momento da geração de uma solução candidata, sendo desconsideradas tanto durante a aplicação dos operadores genéticos quanto na avaliação do indivíduo.

Por outro lado, o número de ações presentes em cada região não têm uma relação com o número de ações presentes na sua respectiva camada do grafo de planos. Esse número é obtido aleatoriamente para cada região do cromossomo em construção. Tal aleatoriedade implica na formação de cromossomos de tamanhos variados, o que exige uma adaptação nos métodos referentes aos operadores genéticos.

Durante o processo evolutivo, os indivíduos são considerados como simples seqüência de ações, tal como feito no algoritmo proposto por Lecheta. A partir de então, serão apresentadas algumas características dos operadores genéticos e da função de aptidão.

Um operador genético, de compressão, foi implementado com a finalidade de retirar do cromossomos aquelas ações as quais não são executadas. Esse operador é executado com uma determinada probabilidade, assim como é feito para a mutação e o cruzamento. Pode-se pensar que tal operação genética determinaria, aos poucos, a diminuição do tamanho de todos os indivíduos da população. Esse problema, no entanto, é naturalmente contornado pela forma com que o operador de cruzamento foi implementado.

Devido à existência de estruturas cromossômicas de tamanhos variados, o operador genético de cruzamento teve de ser adaptado para lidar com os indivíduos. Neste caso, os limites definidos para a geração de pontos de cortes passam a ser diferentes nos indivíduos selecionados para o cruzamento. Essa característica implica que cada um dos filhos gerados pode ter um tamanho menor, igual ou maior do que seus pais. Considerando as implicações dos operadores genéticos de compressão e de cruzamento, percebe-se que enquanto o primeiro tende a diminuir o tamanho dos cromossomos, o segundo permite que este tamanho seja expandido com a introdução de novo material genético.

A função de avaliação utilizada pelo planejador de Castilho e colegas é semelhante à do planejador visto na seção anterior. Um pequeno diferencial é que essa função verifica a executabilidade de ações em dois sentidos: do estado inicial para o objetivo; do objetivo para o estado inicial. A relação de ações executáveis é, então, combinada com a porcentagem de objetivos alcançados.

Os resultados apresentados por este planejador foram muito interessantes, destacando-se principalmente nos domínios *gripper* e *mundo dos blocos*. Seu desempenho pode ser considerado muito bom visto que permitiu resolver problemas os quais somente o *FF* foi capaz de solucionar [21].

4.1.3 *GASat*

O *GASat* [23][24] é um sistema resolvedor de fórmulas SAT baseado em algoritmos genéticos que, em 2004, obteve resultados satisfatórios em uma competição internacional de resolvedores de mesma categoria (SAT'04 Competition). O interesse em investigar esse sistema se deve tanto ao fato de que podem haver determinadas semelhanças entre tratar conflitos em uma rede de Petri e satisfazer cláusulas em uma fórmula SAT, como também é possível traduzir uma instância da classe *petrinet* em uma fórmula SAT.

As fórmulas consideradas neste caso estão na forma normal conjuntiva (CNF - conjunctive normal form), as quais têm como característica ser um conjunção de cláusulas. Uma cláusula é uma disjunção de literais (variáveis, negadas ou não).

A codificação proposta pelo algoritmo determina que cada gene presente em um cromossomo está associado a uma variável da fórmula em análise. Sendo assim, o tamanho de um indivíduo (número de genes) é o número de variáveis distintas que aparecem no conjunto de cláusulas.

Um indivíduo, por sua vez, pode satisfazer a fórmula SAT ou não. No caso de não satisfaze-la, é preciso calcular um valor que represente a qualidade desta solução candidata. A forma adota para isso foi considerar o número de cláusulas satisfeitas. Soluções que satisfazerem mais cláusulas serão consideradas de maior qualidade e, portanto, terão maior probabilidade de serem selecionadas durante o processo evolutivo.

No algorimo proposto pela seção 4.2 será visto que é necessário realizar um tratamento de conflitos para resolver um problema de alcançabilidade em uma rede de Petri. Sendo assim, parte do cálculo da função de avaliação pode ter uma analogia com a função apresenta pelo *GASat*. Na rede de Petri, entretanto, satisfazer todos os conflitos não implica na obtenção de uma solução. Uma solução neste caso necessita que a decisão sobre os conflitos permita, também, uma correta propagação da rede em estudo.

4.2 O algoritmo GAPNet

Esta seção apresenta o algoritmo *GAPNet*, o qual é um sistema planejador em inteligência artificial baseado no paradigma dos algoritmos genéticos. Tal planejador parte de um problema modelado com a rede de Petri como vista na seção 2.5.2 e aplica uma técnica evolucionária para tratar os conflitos existentes na mesma. Tal tratamento permite resolver um problema de alcançabilidade na rede em análise, o que, ao mesmo tempo, significa resolver o problema de planejamento por ela representado.

As próximas seções apresentam tanto as idéias e decisões tomadas com relação ao algoritmo, como também a forma com que as mesmas foram implementadas. Isso engloba motivação (4.2.1), classificação de ações (4.2.2), codificação (4.2.3), geração da população inicial (4.2.4), função de aptidão (4.2.5), otimização no cálculo matricial (4.2.6), corte na rede de Petri (4.2.7), desconsideração das ações de manutenção (4.2.8), elitismo (4.2.9), diversidade (4.2.10), seleção por torneio (4.2.11), operadores genéticos de cruzamento e mutação (4.2.12.1 4.2.12.2), atualização da população (4.2.13), monitoramento da população (4.2.14), novos operadores genéticos (4.2.15), variações da classificação de ações (4.2.16), variações da função de aptidão (4.2.17) e, por fim, a seção 4.2.18 reúne todo esse conteúdo e estrutura o processo evolutivo do planejador proposto.

Os resultados obtidos pelo sistema planejador proposto são apresentados nas seções 4.3 e 4.4.

4.2.1 Motivação

As seções 4.1.1, 4.1.2 e 4.1.3 apresentaram, respectivamente, duas propostas de planejadores baseados em algoritmos genéticos e uma proposta de um resolvedor de fórmulas SAT também baseado neste mesmo paradigma. O algoritmo de Castilho e colegas, por exemplo, mostrou-se bastante promissor ao tratar um problema de planejamento a partir de uma representação do grafo de planos. O *GASat*, por sua vez, foi eficiente ao tratar um conjunto de cláusulas para solucionar um problema de satisfabilidade. Adicionalmente, esse último algoritmo também é útil caso seja feita uma transformação da rede de Petri gerada pela classe *petrinet* em uma instância SAT.

Por outro lado, em ambos trabalhos percebeu-se uma ausência de preocupação com relação a classificar as ações disponíveis em um dado problema. E o que significa a expressão "classificar as ações"? Considerando que um problema de planejamento é caracterizado por encontrar uma seqüência de ações as quais tornam possível a transição entre um estado inicial e um estado objetivo, de que forma cada ação existente influencia nesta solução? Em outras palavras, de que forma pode-se classificar um conjunto de ações de maneira a diferenciá-las de acordo com a importância com que cada uma apresenta sobre um determinado objetivo?

A existência dessa "classificação", além de ser a primeira decisão tomada com relação ao desenvolvimento do algoritmo proposto, pode vir a ser um fator relevante no que diz respeito à análise do espaço de busca e também à convergência da população de um algoritmo genético.

Portanto, acredita-se que o uso de uma técnica evolucionária, incorporada de tal característica, possa ser muito útil ao tratar um problema de planejamento representado por uma rede de Petri. A seção 4.2.2 descreve o procedimento adotada para realizar a classificação de ações.

4.2.2 Classificação de ações

Assim como já mencionado, o *GAPNet* propõe tratar um conjunto de conflitos em uma rede de Petri de forma a solucionar um problema de alcançabilidade na mesma e, por con-

seqüência, resolver um problema de planejamento. Como contribuição, a classe *petrinet* do ambiente IPE fornece um conjunto de pares de ações conflitantes para cada camada de ações presente na referida rede de Petri.

Seja uma rede de Petri N onde cada conflito de ações p está associado a um par de transições (ações) $p = (t, t')$, tem-se um conjunto de pares de ações *mutex* $M = \{(t_1, t_2) \in M \mid \exists p = (t, t')\}$.

Define-se a importância de uma ação t como sendo a influência que a mesma exerce sobre as demais ações do conjunto M. Em outras palavras, significa dizer que a importância de uma ação t está associada ao número de outras ações cuja execução depende, direta ou indiretamente, de t. Para fazer uma classificação de acordo com essa característica, decidiu-se construir um grafo G não dirigido onde:

- cada nodo representa uma ação distinta presente no conjunto M;
- cada aresta representa um conflito entre dois nodos (ações).

Visto que o conjunto M possui conflitos relativos às diferentes camadas de ações de uma rede de Petri, o grafo G será não conexo. Cada componente conexa de G representará uma camada de ações da rede. Isso ocorre porque ações de uma camada não possuem conflitos diretos com ações de outras camadas, embora possam influenciar indiretamente.

Sendo assim, a importância de uma ação t perante o conjunto M pode ser encontrada a partir de G e de um parâmetro $prof$, conforme descreve o algoritmo 5.

Algoritmo 5 Cálculo de importância de ações

1: A partir de G, selecione a componente conexa G' na qual t está presente;
2: Crie uma uma lista I a qual contenha todas as ações distintas alcançáveis a partir de t, onde o número máximo de arestas percorridas deve ser igual a $prof$;
3: Retorne o número de elementos de I.

Uma vez executado o algoritmo 5, realiza-se um ordenação das ações em ordem crescente de seus valores de importância. Como resultado, tem-se uma lista de ações onde as primeiras são consideradas menos importantes e, as últimas, mais importantes. Neste caso, temos uma classificação global de ações levando-se em conta a influência de cada uma com relação à sua respectiva camada.

Por outro lado sabemos que cada ação, além de influenciar diretamente ações de sua mesma camada, influencia indiretamente ações de camadas vizinhas. Esse detalhe implicou na criação de um parâmetro que determina se a "importância" de uma ação deve ou não deve ser propagada para as próximas camadas.

Essa propagação determina que a importância de uma ação t será representada pelo acúmulo de importâncias de ações as quais poderão ser executadas posteriomente a ela. Com essa decisão, ações presentes nas primeiras camadas da rede tenderão a ser mais valorizadas. Mais adiante serão apresentadas algumas variações dessa classificação.

Por fim, independente do uso ou não da propagação, a lista com os valores de importância obtidos será usada como base para definir um cromossomo, conforme mostrado na seção 4.2.3.

4.2.3 Codificação

O planejador *GAPNet*, ao contrário daqueles vistos nas seções 4.1.1 e 4.1.2, utiliza uma representação binária nos seus cromossomos, onde cada gene está vinculado a uma determinada ação da rede de Petri gerada pela classe *petrinet* do ambiente IPE. Enquanto que um valor verdadeiro implica na execução da ação, um valor falso implica na não execução da ação.

O número de genes em um cromossomo é n, onde n é o número de ações distintas presentes nos pares de conflitos do conjunto M da seção 4.2.2. Internamente, as ações são ordenadas em ordem crescente do grau de prioridade (importância) obtido pela classificação das ações. Sendo assim, quanto mais à direita estiver uma ação no cromossomo, mais signficativa a mesma será.

A idéia por traz dessa classificação é diferenciar as ações de acordo com a dificuldade de satisfazer o conjunto de conflitos. Dessa forma, espera-se que seja mais vantajoso resolver primeiro os conflitos que envolvem ações de alta prioridade e, por último, os demais. Pode-se pensar que decidir sobre os conflitos menos impactantes na rede é o mesmo que realizar pequenos ajustes para encontrar uma solução, tarefa a qual pode ser executada posteriomente.

4.2.4 Geração da população inicial

A seção 4.2.3 mostrou que um indivíduo é representado por uma lista de ações (transições) conflitantes da rede de Petri em análise. Os valores possíveis para cada gene do cromossomo são "1" para execução e "0" para não execução.

Para formar a população inicial foi implementado, inicialmente, um método o qual gera aleatoriamente valores binários para cada indivíduo. Dessa forma, considerando que a probabilidade de gerar o bit "0" ou "1" é de 50%, a tendência é que um cromossomo contenha metade dos seus genes habilitadados para execução (1).

No entanto, ao analisar as primeiras fases de testes, foi constatado que, nos problemas considerados, os indivíduos convergiam para soluções nas quais predominavam a presença de valores "0" em seus genes. Isso é justificado pelo fato de que na maioria dos problemas poucas ações são executadas em paralelo, significando que para cada camada de ações serão selecionas apenas algumas transições.

Visando acelerar o processo de convergência, foi realizada uma adaptação no método de geração de indivíduos. Tal modificação consiste em definir (aleatoriamente) quantos genes do cromossomo serão marcados como "1". Para isso defini-se que o menor número de ações executadas será igual ao número de camadas de ações e o maior número de ações executáveis será o tamanho do indivíduo.

Dessa forma, gera-se um número aleatório entre os limites inferior e o superior previamente definidos. Considerando g como sendo o valor obtido, faz-se o seguinte: Seleciona-se aleatoriamente g posições do cromossomo e, para cada uma, marca-se o gene com o valor "1".

Essa alternativa permite que algumas soluções candidatas sejam formadas por muitas ações executáveis, entretanto, também possibilita que outras sejam formadas por poucas ações executáveis. Portanto, essa característica gera uma população mais diversa, abrangendo regiões mais heterogêneas do espaço de busca.

4.2.5 Função de aptidão

Está seção apresenta a função implementada para definir a qualidade de cada solução (indivíduo) com relação ao problema.

Inicialmente deve-se notar que, conforme a codificação adotada, nem toda associação de valores para um cromossomo virá a ser uma seqüência de ações válida. Dessa forma, a função de aptidão foi dividida em duas principais sub-funções: verificação da consistência de um cromossomo e dificuldade para obtenção de objetivos.

A verificação da consistência é responsável por analisar se os valores associados aos genes (ações) de um cromossomo satisfazem a relação de conflitos do conjunto M da seção 4.2.2. Considerando que o conjunto M possui m conflitos, esta função devolve um valor entre 0 (zero) e m, o qual indica quantos conflitos foram resolvidos.

Já a dificuldade para obtenção de objetivos visa determinar as barreiras encontradas para alcançar o estado objetivo. O que este procedimento faz é utilizar as equações matriciais do formalismo de redes de Petri para disparar as ações setadas pelo cromossomo. A partir disso é feita uma suposição de que todos os sub-objetivos foram satisfeitos. Após, realiza-se uma busca nos lugares da rede de Petri encontrando locais onde a marcação indica que a suposição escolhida é falsa. O valor devolvido por essa função está entre 0 (zero) e o número de lugares p existentes na rede de Petri.

Ao evoluir um conjunto de soluções candidatas o algoritmo está, na verdade, maximizando a consistência do cromossomo e minimizando a dificuldade de obtenção de objetivos. Uma solução obrigatoriamente deverá conter um valor de dificuldade igual a zero, o que só é possível com consistência máxima. Vale ressaltar que a recíproca não é verdadeira, ou seja, um caso de consistência máxima não necessariamente exige dificuldade de obtenção de objetivos igual a zero.

A relação de não equivalência entre essas duas funções existe pelo seguinte motivo: um conflito é um par de ações e, para resolvê-lo, basta não executar as duas ações concomitantemente. Porém, imagine que para alguns conflitos a decisão tomada seja não executar nenhuma das ações envolvidas. Do ponto de vista da resolução de conflitos isso está logicamente correto, pois não faz sentido obrigar que para todo conflito pelo menos

uma ação deva ser executada. Isso poderia fazer com que um plano na rede nunca fosse encontrado. Entretanto não executar nenhuma das ações também é problemático, visto que uma das ações pode ser essencial para obtenção da solução.

Uma vez que o cálculo de consistência do cromossomo é de certa forma simples, será detalhado somente o cálculo do valor da dificuldade de obtenção de objetivos. Durante o cálculo da função de aptidão estão disponíveis as seguintes informações:

- relação de quais ações devem e não devem ser executadas, obtida a partir da valoração do cromossomo;
- uma instância da rede de Petri a qual modela o problema de planejamento em questão.

A seção 2.5.2 mostrou as características da rede de Petri gerada pela classe *petrinet*. Foi visto que a rede, além de alternar entre camada de ações e literais, apresenta um padrão a cada quatro camadas: a primeira delas (L_0) possui os literais propriamente ditos que constituem um conjunto de possíveis estados; a segunda (L_1) possui transições as quais são responsáveis por fazer cópias dos literais presentes na primeira; a terceira (L_2) possui as cópias geradas pela segunda camada; já a quarta e última (L_3) é formada pelas possíveis ações a serem executadas.

As ações (transições) que são consideradas durante a codificação do cromossomo são aquelas que, além de pertencerem à camada de padrão L_3, possuem pelo menos um conflito. Sendo assim, percebe-se que existe um grupo de ações as quais nunca aparecem no cromomossomo. Esse grupo é formado pelas transições: pertencentes à camada de padrão L_1; e as pertencentes à camada de padrão L_3 que, ao mesmo tempo, são livres de conflitos.

O algoritmo o qual será apresentado posteriormente, referente a propagação da rede de Petri, possui as seguintes características:

- a propagação da rede é feita para cada camada de transições (L_1 e L_3) existente na rede;

- quando a camada pertencer ao padrão L_1, serão executadas todas as transições as quais possuem suas pré-condições satisfeitas pela rede;
- quando a camada pertencer ao padrão L_3: se a ação pertence ao cromossomo, sua executabilidade será definida pela valoração do mesmo, caso contrário, será executada somente se suas pré-condições forem satisfeitas pela rede.
- as equações matriciais do formalismo de rede de Petri são aplicadas para obter a nova marcação da rede;
- a execução incorreta de algumas ações (marcadas pelo cromossomo) resultará em marcações negativas, o que servirá de informação para o cálculo da função de aptidão;
- o vetor característico possui informações de quais transições serão disparadas e é usado no cálculo matricial.

De forma mais estruturada, o algoritmo 6 descreve como que uma rede de Petri N é propagada para avaliar uma solução candidata S.

A execução do algoritmo de propagação devolve uma marcação final M da qual pode-se retirar algumas informações sobre a solução avaliada. Para facilitar essa análise subtrai-se de M a marcação objetivo (final) M_f, obtendo-se uma marcação M'. A partir de M' pode-se inferir o seguinte:

- Se um lugar mutex estiver com marcação negativa, significa que o mesmo não foi satisfeito;
- Se um lugar objetivo estiver com marcação negativa, significa que o mesmo não foi alcançado;
- Se um outro lugar qualquer estiver com marcação negativa, significa que o mesmo foi uma barreira durante a propagação da rede.

Embora o valor de consistência de um cromossomo possa ser obtido diretamente a partir da função de cálculo de dificuldades, foi mantido um procedimento em separado

Algoritmo 6 Propagação da rede de Petri N a partir de uma solução candidata S

1: Marque a rede N de acordo com sua marcação (estado) inicial; {Não esquecer que, assim como descrito na seção 2.5.2, todos os mutex apresentam também marcação inicial igual a um.}
2: Crie uma lista LC contendo uma identificação para todas as camadas de padrão L_1 e L_3, organizadas na ordem na qual aparecem na rede.
3: Zere o vetor característico;
4: **for all** $C \mid C \in LC$ **do**
5: **if** (C é do padrão L_1) **then**
6: **for all** $t \mid t \in C$ **do**
7: **if** (t é executável) **then**
8: marque t como "1" no vetor característico;
9: **else**
10: marque t como "0" no vetor característico;
11: **else**
12: **for all** $t \mid t \in C$ **do**
13: **if** $t \in S$ **then**
14: marque a executabilidade de t de acordo com o valor encontrado no cromossomo S;
15: **else**
16: **if** (t é executável) **then**
17: marque t como "1" no vetor característico;
18: **else**
19: marque t como "0" no vetor característico;
20: Propague a rede de Petri utilizando o vetor característico;
21: Zere o vetor característico;

para realizar tal tarefa. Isso deve-se ao fato de que quando os métodos relacionados a essa característica foram implementados, não estava-se fazendo uma completa distinção dos lugares os quais apresentavam marcação negativa na rede de Petri. Adicionalmente, pode-se calcular a consistência de uma solução candidata sem a necessidade de propagar a rede de Petri.

A função que calcula a dificuldade de obtenção de objetivos devolve o número total de lugares cuja marcação resultante foi negativa. Sendo assim, a primeira abordagem definida para ser a função de aptidão é: quanto maior o número de lugares satisfeitos na rede de Petri, mais bem qualificada será considerada a solução em análise. Posteriomente, na seção 4.2.17, serão descritas alternativas para esta abordagem.

As três próximas seções descrevem algumas otimizações relacionadas à função de aptidão.

4.2.6 Otimização no cálculo matricial

O processo utilizado para cálculo do valor de aptidão é bastante custoso, principalmente pelo fato da necessidade de executar um cálculo matricial para cada camada de transições presente na mesma.

Com a intenção de reduzir parcialmente o tempo de processamento dessa função, foi implementado um método que reconhece qual região da rede de Petri está sendo propagada e permite que o cálculo matricial limite-se a esta mesma região. Isso é possível a partir de uma identificação da primeira e da última transição da camada em análise.

A marcação obtida para uma rede de Petri após o disparo de uma seqüência de transições pode ser calculada pela equação 2.2. Entretanto, ao disparar transições de apenas uma camada, boa parte do vetor característico $\overline{s}$ conterá valores iguais a zero. Tais valores anularão os valores da matriz de incidência C, significando que esses últimos podem ser desconsiderados no cálculo de atualização da marcação.

Sendo assim, a equação fundamental foi adaptada para fazer um produto matricial apenas de uma parte da matriz de incidências C e do vetor característico $\overline{s}$. O vetor $\overline{s}$ é limitado pelo menor e pelo maior índice das transições presentes na camada considerada.

Já a matriz de incidências é limitada inferiormente pelo menor índice de lugar p o qual é pré-condição de uma das transições disparadas e, superiormente, pelo maior índice de lugar p o qual é efeito de uma das transições disparadas.

O vetor resultante do produto matricial tem dimensão igual ao número de lugares considerados na matriz de incidências ($maior\_indice - menor\_indice + 1$). Entretanto, ele deve ser expandido para possuir dimensão igual ao número total de lugares p existentes na rede. Com essa finalidade, cada lugar não considerado é adicionado neste vetor em sua posição correta (de acordo com seu índice) e com valor igual a zero, significando que neste lugar não haverá modificação da marcação. Finalmente, este vetor é somado ao da marcação atual, resultando uma nova configuração na rede de Petri.

Portanto, esse método tem como objetivo otimizar o cálculo matricial necessário para propagação da rede. Seu princípio é limitar o produto entre matriz de incidências e vetor característico à uma região específica, determinada pelas transições da camada em análise e pelos lugares envolvidos nos respectivos disparos.

4.2.7 Corte na rede de Petri

Uma característica da rede de Petri gerada pela classe *petrinet*, que também existe no grafo de planos, é a possibilidade de executar um corte nas últimas camadas geradas pelo algoritmo de expansão.

Ao analisar as últimas camadas da rede, nota-se a existência de vários lugares (características) os quais não fazem parte do estado objetivo. A presença de tais lugares está correta com relação à lógica de expansão realizada, entretanto, os mesmos podem ser suprimidos sem implicar problemas para o processo de busca por uma solução, conforme é descrito no algoritmo 7.

É preciso ressaltar que, após a execução do algoritmo 7, haverá alterações no conjunto M de conflitos descrito pela seção 4.2.2. Sendo necessário que o mesmo seja revisto, excluindo aqueles conflitos os quais envolvem transições descartadas durante o processo de corte na rede de Petri. Na versão implementada pelo *GAPNet*, esta verificação é feita já durante a execução do algoritmo 7.

Algoritmo 7 Procedimento de corte na rede de Petri

1: Crie uma lista *prop* contendo os literais do estado objetivo;
2: L recebe a última camada de lugares pertencente ao padrão L_0
3: Crie uma lista vazia *trans*
4: **while** $(prop! = L)$ **do**
5: **for all** $l \mid l \in L$ **do**
6: **if** $(l \in prop)$ **then**
7: Insira em *trans* as transições que geram l;
8: **else**
9: Desfaça as ligações de l com as transições que a geram;
10: Elimine l;
11: **if** (L é a primeira camada da rede) **then**
12: **return** 1
13: T recebe a camada de transições antecedente à C;
14: Limpe a lista *prop*
15: **for all** $t \mid t \in T$ **do**
16: **if** $(t \in trans)$ **then**
17: Insira em *prop* as pré-condições de t;
18: **else**
19: Desfaça as ligações de t com suas pré-condições;
20: Elimine t;
21: Limpe a lista *trans*;
22: L recebe a camada anterior à T;
23: **return** 1

4.2.8 Desconsiderando ações de manutenção

Essa seção faz uma análise mais completa do tipo de transições presente na rede de Petri, buscando diminuir o tamanho do cromossomo a partir de uma "desconsideração" das ações de manutenção. Até o momento, um cromossomo é formado por ações conflitantes pertencentes a dois grupos:

- ações de manutenção;
- ações que não são de manutenção.

Assim como visto na seção 4.2.3, o método de codificação não diferencia estes dois tipos de ações. Porém, com a definição de um novo parâmetro, é possível permitir que o algoritmo *GAPNet* desconsidere as ações de manutenção durante a geração de soluções candidatas.

Isso é feito da seguinte maneira:

- Identifica-se quais ações do cromossomo são de manutenção;
- Separa-se o cromossomo em duas grandes regiões, fazendo-se com que a primeira contenha apenas ações de manutenção;
- Desconsidera-se os conflitos que estão relacionados às ações de manutenção;
- Redimensiona-se o cromossomo, para que o mesmo seja constituído somente pela segunda região;
- Mantém-se uma cópia da primeira região;
- Ordena-se as ações do cromossomo de acordo com as definições das seções 4.2.2 e 4.2.3

Com essa modificação o cromossomo passa a conter somente ações conflitantes e que, ao mesmo tempo, não são de manutenção. Isso facilita o processo evolutivo visto que o espaço de busca a ser considerado é reduzido. É evidente, também, que durante a função

de avaliação, as ações de manutenção deverão ser consideradas para que a rede de Petri seja corretamente propagada.

Quando as ações de manutenção conflitantes são desconsideradas na representação do cromossomo, a função de propagação deve ser adaptada, assim como descreve o algoritmo 8.

Algoritmo 8 Propagação da rede de Petri, codificação sem ações de manutenção

1: Marque a rede N de acordo com sua marcação (estado) inicial; {Não esquecer que, assim como descrito na seção 2.5.2, todos os mutex apresentam também marcação inicial igual a um.}
2: Crie uma lista LC contendo uma identificação para todas as camadas de padrão L_1 e L_3, organizadas na ordem na qual aparecem na rede.
3: $S = cromossomo$;
4: Ma = ações de manutenção; {O vetor Ma contém as ações de manutenção que são conflitantes e que faziam parte do cromossomo.}
5: Crie uma lista vazia *usedactions*;
6: **for all** $C \mid C \in LC$ **do**
7: **if** (C é do padrão L_1) **then**
8: **for all** $t \mid t \in C$ **do**
9: **if** (t é executável) **then**
10: marque t como "1" no vetor característico;
11: **else**
12: marque t como "0" no vetor característico;
13: **else**
14: **for all** $t \mid t \in C$ **do**
15: **if** $t \in S$ **then**
16: marque a executabilidade de t de acordo com o valor encontrado no cromossomo S;
17: caso tenha sido marcada como executável, adicione t em *usedactions*;
18: **else**
19: **if** ($t \ni Ma$) **then**
20: **if** (t é executável) **then**
21: marque t como "1" no vetor característico;
22: **else**
23: marque t como "0" no vetor característico;
24: **for all** $t \mid t \in Ma$ **do**
25: **if** ((t é executável) && (não conflitante(t,*usedactions*))) **then**
26: marque t como "1" no vetor característico;
27: insira t em *usedactions*;
28: **else**
29: marque t como "0" no vetor característico;
30: Propague a rede de Petri utilizando o vetor característico;

A diferença deste algoritmo para o apresentado anteriormente (6), é que é dado prioridade para as ações que não são de manutenção. Dessa forma, as primeiras ações a serem marcadas na rede de Petri serão as conflitantes (que não são de manutenção) e, posteriormente, as não conflitantes. Somente após marcar tais ações, é que será verificado quais das de manutenção (conflitantes) poderão ser executadas.

4.2.9 Elitismo

O método referente ao elitismo cria uma cópia dos melhores indivíduos existentes na geração atual. O número de indivíduos a serem selecionados é definido pelo parâmetro *taxa de elitismo*, o qual é um percentual do tamanho da população.

Seu propósito é garantir que os melhores genes da geração atual estejam presentes na próxima geração. Seu uso depende de como está sendo realizada a atualização da população, que será vista na seção 4.2.13.

4.2.10 Diversidade

Um problema que eventualmente surge, dependendo da forma com que se realiza a atualização da população, é que não existem indivíduos suficientes para completar (sem repetição) o tamanho correto da nova população.

Uma solução para este problema é gerar aleatoriamente um grupo de indivíduos e adicioná-los a população, porém, esse procedimento provoca uma oscilação muito grande nos valores de aptidão entre duas gerações. Como alternativa, o método da diversidade gera um pequeno grupo de indivíduos que representam parcialmente o espaço de busca abrangido pela população atual, com exceção daquela parte representada pelos indivíduos pertencentes à elite.

Embora tal método tenha sido criado para ser usado em situações onde é necessária a inserção de novos indivíduos na população, o mesmo tornou-se uma nova alternativa de implementação da atualização de população, conforme será visto na seção 4.2.13.

4.2.11 Seleção por torneio

A técnica escolhida para realizar a seleção de indivíduos, os quais serão submetidos ao operador de cruzamento, foi a seleção por torneio.

Enquanto o parâmetro *tamanho torneio* define o número de indivíduos que participarão de um torneio, o parâmetro *taxa de cruzamento* define o número de torneios a serem executados. Tanto *tamanho do torneio* quanto *taxa de cruzamento* são um percentual do tamanho da população.

Os indivíduos selecionados são copiados para uma população temporária P', onde mais tarde será efetivada a recombinação genética por meio do operador de cruzamento

4.2.12 Operadores genéticos

Nesta seção são apresentadas as implementações de cada operador genético no planejador *GAPNet*.

4.2.12.1 *Cruzamento*

O operador de cruzamento foi implementado com duas variações. Enquanto a primeira (cruzamento 1) utiliza um ponto de corte no cromossomo, a segunda (cruzamento 2) utiliza dois pontos de corte. O parâmetro que define qual das variações deve ser utilizada é o *tipo de cruzamento*.

Em muitos casos, os indivíduos selecionados para serem submetidos ao operador de cruzamento apresentam alguma semelhança genética. Tal semelhança tende a aumentar a medida que a população evolui, ou seja, aos poucos o número de genes iguais entre dois indivíduos aumenta.

Uma vez que o objetivo do operador de cruzamento é recombinar o material genético de dois indivíduos, não faz sentido gerar filhos idênticos aos seus progenitores. Essa situação pode ocorrer de acordo com as semelhanças entre dois indivíduos e o ponto de corte selecionado.

Considere os dois cromossomos mostrados na imagem 3.1. Note que os três genes mais

a direita de cada um deles, possuem valores iguais. Neste caso, para valores de pontos de corte nesta região, os cromossomos gerados terão características idênticas às dos seus pais. Para evitar isso, o método de cruzamento desenvolvido implementa uma verificação de semelhanças entre cromossomos, determinando um intervalo válido para a geraração de pontos de corte. Essa verificação contribui tanto para uma evolução mais eficiente da população, quanto também para evitar manipulação desnecessária de material genético.

4.2.12.2 *Mutação*

A mutação também apresenta duas variações de implementação, onde a diferença entre elas está ligada a forma com que é considerado o parâmetro *taxa de mutação*. Tal parâmetro pode ser encarado como um percentual do tamanho da população ou como um percentual do total de genes presente na população.

A diferença é que, quando considerado como um percentual da população, a taxa de mutação está fazendo referência a quantos indivíduos devem sofrer mutação a cada geração (mutação 2). No caso de ser considerada como um percentual de genes da população, a taxa de mutação indica quantos genes sofrerão mutação a cada geração (mutação 1).

Embora as duas abordagens possam parecer muito semelhantes o que acontece é que no segundo caso o número de genes alterados é muito maior, mesmo quando considerada uma taxa de mutação equivalente para ambas. Porém como o intuito da mutação é, além de garantir diversidade, propiciar pequenos ajustes nas soluções exploradas, a opção a qual considera o "número de indivíduos" tem sido usada com maior freqüência.

Assim como o operador de cruzamento, a mutação também salva seus resultados em uma população temporária P'.

4.2.13 Atualização da população

Assim como a maioria dos métodos implementados, a atualização da população também apresenta algumas variações visando obter diferentes análises da evolução de uma população. Inicialmente será apresentado o conjunto de informações o qual fica disponível para esse método e, posteriormente, será visto de que forma essas informações podem ser

utilizadas.

Antes do método de atualização da população ser executado, estão disponíveis os seguintes dados:

- *Elite*: conjunto dos melhores indivíduos da população P, calculado a partir de uma taxa de elitismo;
- *Diversos*: conjunto de indivíduos da população P, selecionados aleatoriamente a partir de uma taxa de diversidade;
- P: população original desta geração;
- P': população gerada a partir dos operadores de cruzamento e de mutação.

De que forma esses quatro conjuntos *Elite*, *Diversos*, P e P' podem ser combinados a fim de formar uma nova população P de tamanho padrão n?

A primeira opção é mostrada pelo algoritmo 9.

Algoritmo 9 Atualização da população - opção 1

1: Avalie P';
2: Limpe a população P;
3: Adicione os indivíduos de *Elite* em P;
4: Ordene os indivíduos de P' em ordem crescente de aptidão;
5: Enquanto o tamanho de P for diferente de n e o tamanho P' for diferente de zero, retire o melhor indivíduo de P' e insira em P;
6: Caso o tamanho de P seja menor que n, gere aleatoriamente indivíduos para completar o tamanho de P;
7: Retorne P.

Alternativamente, para evitar grandes oscilações de aptidão causadas pelo passo 6, o mesmo pode ser substituído por: Caso o tamanho de P seja menor que n, complete P com os indivíduos presentes em *Diversos*.

Uma segunda opção realiza os passos descritos pelo algoritmo 10

Algoritmo 10 Atualização da população - opção 2

1: Avalie P'
2: Adicione os indivíduos de P' em P;
3: Ordene P em ordem decrescente de aptidão;
4: Retorne P contendo apenas seus n primeiros indivíduos.

A grande diferença entre essas duas primeiras implementações é o uso ou não da população P. Tal característica é habilitada pelo parâmetro *usar população anterior*. Quando este parâmetro tem valor verdadeiro, os parâmetros de elitismo e diversidade devem ser setados com percentual igual a zero.

Assim como será visto na seção de resultados, a convergência da população ocorre de forma bem mais rápida quando utiliza-se a segunda implementação, porém, em alguns problemas tratados, isso não significou uma vantagem para obtenção da solução.

Buscando formar um processo de evolução intermediário entre as duas opções apresentadas, também foi proposta uma terceira implementação a qual utiliza o parâmetro de diversidade de uma forma um pouco modificada.

Analisando melhor a primeira implementação, percebe-se que os indivíduos presentes em *Diversos* são utilizados de vez em quando e também em quantidades variadas. Sendo assim, propõe-se uma alternativa a qual utiliza a cada geração uma porção fixa de indivíduos diversificados. Veja o algoritmo 11.

Algoritmo 11 Atualização da população - opção 3

1: Avalie P';
2: Limpe a população P;
3: Adicione os indivíduos de *Elite* em P;
4: Adicione os indivíduos de *Diversos* em P;
5: Ordene os indivíduos de P' em ordem crescente de aptidão;
6: Enquanto o tamanho de P for diferente de n e o tamanho P' for diferente de zero, retire o melhor indivíduo de P' e insira em P;
7: Caso o tamanho de P seja menor que n, gere aleatoriamente indivíduos para completar o tamanho de P;
8: Retorne P.

Desta forma, assim como na primeira implementação, pode haver uma oscilação devido a ocorrência do passo 7. No entanto, neste caso, as chances disso ocorrer são menores e, de qualquer forma, pode ser contornada por um ajuste do parâmetro de diversidade.

4.2.14 Monitoramento da população

Muitas vezes, durante a execução do planejador, pode-se detectar algumas características que indicam uma má evolução das soluções candidatas. Visando contornar alguns proble-

mas dessa natureza, foram adicionados alguns parâmetros para monitorar a população e, eventualmente, realizar alguns ajustes.

Foi decidido implementar dois tipos principais de ações a serem tomadas. Umas delas diz respeito a modificar dinamicamente algum dos parâmetros e outra de inserir novo material genético na população, ambas sendo executadas caso seja identificada uma estagnação na evolução da mesma.

O parâmetro *limite de não evolução* é usado como um limiar para o disparo das ações de ajustes, e representa o número de gerações nas quais o melhor indivíduo da população não apresentou melhora de aptidão. Uma vez atingido esse limiar, dispara-se uma ação de acordo com o parâmetro *tipo de monitoração*.

Caso este parâmetro esteja setado como *reiniciar população*, a população será reiniciada conforme o método padrão para geração de soluções candidatas, preservando apenas os indivíduos pertencentes ao grupo de elite. Por outro lado, se o parâmetro estiver setado como *reajuste de parâmetros*, os parâmetros genéticos de mutação e cruzamento são elevados consideravelmente durante uma geração.

Essas duas alternativas visam, respectivamente, contornar uma possível estagnação da população realizando através de uma renovação parcial da população e forte indução na ocorrência de combinações/variações genéticas.

4.2.15 Novos operadores genéticos

Esta seção apresenta dois novos operadores genéticos desenvolvidos com a finalidade de melhor interagir com os aspectos dos problemas de planejamento considerados. O primeiro deles é apresentado pela seção 4.2.15.1 e é uma variação da mutação clássica, visando considerar os conflitos existentes na rede de Petri. O segundo operador, descrito pela seção 4.2.15.2, é responsável por reorganizar as ações do cromossomo de acordo com seus significados diante de cada camada existentes na rede.

4.2.15.1 Operador de mutação baseado em conflitos

Ao contrário do operador clássico de mutação que altera uma gene qualquer do cromossomo, este operador modifica ações (genes) em função de algum conflito pertencente ao conjunto M visto na seção 4.2.2. Neste caso, o parâmetro *taxa de mutação* determina a porcentagem de indivíduos da população os quais sofrerão a mutação.

Para cada indivíduo selecionado aleatoriamente, também seleciona-se de forma aleatória um conflito do conjunto M. A partir desse conflito, faz-se uma análise das duas ações evolvidas no mesmo, modificando-as da seguinte forma:

- Caso as duas ações estejam setadas para execução, desabilite aleatoriamente uma delas;
- Caso nenhuma delas esteja habilitada, habilite aleatoriamente uma delas;
- Caso uma esteja habilita e outra não, troque a execução de uma pela outra.

Essa forma de realização da mutação pretende induzir uma variação genética de forma mais adequada, uma vez que tais mudanças levam em consideração restrições existentes na rede. Em outras palavras, esta implementação visa reduzir um pouco a aleatoriedade do operador clássico, introduzindo uma peculiaridade específica do problema em questão.

4.2.15.2 Operador de reorganização de ações

A característica envolvida na criação deste operador diz respeito a uma propriedade presente tanto no grafo de planos quanto na rede de Petri gerada pela classe *petrinet*.

Durante a expansão dessas estruturas é inserido, a cada par de camadas, um novo conjunto de possíveis ações a serem executadas. Entretanto, uma vez adicionada uma ação, a mesma aparecerá em todas as camadas subseqüentes, significando que a mesma pode ser executada em vários pontos da estrutura considerada.

Analisando especificamente o caso das redes de Petri, isso implica na existência de diversas transições diferentes que, no entanto, representam uma mesma ação. Na codificação utilizada por este planejador, é feita uma abstração desse detalhe.

Após verificar resultados insatisfatórios para alguns problemas, percebeu-se a dificuldade de setar no cromossomo a transição correta para disparar uma determinada ação. Em outras palavras significa que em muitos casos a ação correta é executada, porém, em um ponto inadequado do plano (uso de uma transição errada).

Dessa forma, esse operador tem por finalidade modificar a posição de execução de uma ação na rede de Petri, ou seja, procurar por outra transição que represente a mesma ação porém em uma camada diferente. Na implementação realizada este método procura priorizar a escolha de transições presentes nas camadas iniciais da rede, conforme é descrito no algoritmo 12.

Algoritmo 12 Reorganização de ações no cromossomo

1: Salve em C todas as transições marcadas como "1" no cromossomo original;
2: Zere todo o cromossomo original;
3: Ordene as transições de C de acordo com seu aparececimento na rede de Petri. Transições da primeira camada, da segunda e ainda por diante;
4: **for all** $t \mid t \in C$ **do**
5: identifique em cada camada da rede uma transição correspondente a t, ou seja, aquela que representa a mesma ação que t;
6: selecione a primeira transição que não esteja em conflito com as já marcadas no cromossomo original;
7: marque esta transição como "1" no cromossomo original;
8: Identifique as camadas da rede onde nenhuma transição está marcada no cromossomo original;
9: Marque aleatoriamente no cromossomo original algumas transições pertencentes às camadas identificadas pelo passo anterior.

Percebe-se, assim, duas características no algoritmo 12:

- as ações, sempre que possível, são realocadas para transições presentes em camadas anteriores a da original;
- ao final, marca-se aleatoriamente algumas transições para aquelas camadas que não foram utilizadas.

Espera-se que essa reorganização de transições dentro do cromossomo permita uma melhor evolução do conjunto de soluções candidatas, uma vez que foram identificadas certas dificuldades quando ações corretas são representadas por transições erradas na rede de Petri.

4.2.16 Variações da classificação de ações

Conforme visto na seção 4.2.2, as ações podem ser classificadas a partir de um grau de importância (influência) calculada para cada uma e, adicionalmente, por uma propagação dessas influências. Esta seção, por outro lado, propõe organizar as ações considerando suas influências e as camadas envolvidas.

Com essa finalidade, foi definido um parâmetro *Prioridade dentro de camadas*. Assim como seu próprio nome já indica, a idéia é fazer com que as prioridades calculadas tenham relevância apenas no escopo da camada considerada. Isso significa dizer que as ações serão inicialmente ordenadas de acordo com suas camadas e, posteriormente, em relação às suas prioridades.

Para ficar mais claro, pode-se descrever essa classificação da seguinte forma:

- Separe as ações em grupos, de acordo com suas camadas;
- Ordene os grupos em ordem crescente do índice de sua respectiva camada;
- Ordene as ações internas de cada grupo em ordem decrescente de influências;
- Utilize a ordem de ações resultante para a codificação do cromossomo.

Portanto, ao utilizar esta abordagem, está-se dizendo que ações pertencentes às camadas iniciais são mais importantes do que as presentes nas camadas finais. Para aquelas que pertencem a uma mesma cada, realiza-se um desempate de acordo com a importância de cada uma, comforme calculado pela seção 4.2.2.

4.2.17 Variações da função de aptidão

Embora a idéia principal a ser utilizada como função de aptidão tenha sido apresentada na seção 4.2.5, esta seção propõe algumas variações, considerando o número de conflitos satisfeitos na rede e também usando uma ponderação para os lugares da mesma.

Originalmente, a função de aptidão retorna o número total de lugares satisfeitos na rede de Petri, sem realizar nenhuma distinção dos mesmos. Essa função ficou definida

como sendo a funçao de aptidão 1. A seguir são apresentadas duas alternativas para calcular a qualidade de uma solução candidata.

A função de aptidão 2 leva em consideração, também, o número de conflitos satisfeitos na rede de Petri após a propagação das ações setadas no cromossomo. Dessa forma, a mesma é definida como: se o número de lugares não satisfeitos na rede for igual a zero, então retorne o número de mutex's existentes mais um ($nro\_de\_mutex + 1$); caso contrário, retorne o quociente entre o número de mutex's satisfeitos e o número de lugares não satisfeitos ($nro\_mutex\_satisfeitos/nro\_lugares\_nao\_satisfeitos$).

Por último, a função de aptidão 3 realiza uma ponderação dos lugares existentes na rede de Petri, fazendo com que cada um receba uma importância de acordo com a camada a qual o mesmo pertence. No caso implementado, os lugares pertencentes as camadas iniciais tem peso maior do que aqueles das camadas finais. A diferença desta função com relação a função 1 é a ponderação dos lugares na rede.

Posteriormente, na seção de resultados finais (4.4), será visto que a função de aptidão 2 apresentou um comportamento mais adequado durante a evolução da população.

4.2.18 Estruturando o algoritmo GAPNet

Até então este capítulo apresentou diversos aspectos com relação a implementação do algoritmo *GAPNet*. Agora pretende-se descrever como tais informações, em conjunto com a base apresentada no capítulo 2, podem ser agrupadas para constituir o algoritmo como um todo.

O início do sistema planejador consiste em utilizar algumas classes disponibilizadas pelo ambiente IPE com o intuito de modelar um problema de planejamento. Na seção 2.4 há uma descrição de como obter uma representação baseada em redes de Petri, a qual servirá de entrada para o resolvedor *GAPNet*.

Após instanciar um objeto da classe *gapnet*, é necessário chamar o método *solve()*, o qual é responsável por começar o processo evolutivo em busca de uma solução. Caso o sistema planejador encontre um plano este método retorna *verdadeiro*, caso contrário *falso*. Os principais procedimentos executados por esse método são descritos abaixo:

- Inicialização dos dados, que engloba: definição das marcações inicial e objetivo; formação da matriz de incidências, corte na rede de Petri; identificação do conjunto de mutex; classificação de ações; geração da população inicial;
- Avaliação da população;
- Armazenamento de dados da primeira geração;
- Execução do processo evolutivo;

O processo evolutivo é brevemente descrito pelo algoritmo 13.

Algoritmo 13 Processo evolutivo

```
1: geracao_atual = 0;
2: while (geracao_atual <= Max_gen)||(plano_nao_encontrado) do
3:    Monitoramento();
4:    Elite();
5:    Selecao();
6:    Cruzamento();
7:    Mutacao();
8:    Reorganizacao();
9:    Atualizacao_Populacao(); {Inclui reavaliação de indivíduos}
10:   Armazenamento de dados da geração;
11:   Checagem de monitoramento;
12:   if (encontrou um plano) then
13:      return resolvido;
14:   geracao_atual++;
15: return não resolvido;
```

Quando o método *solve()* não encontra um plano, duas situações são possíveis: um plano não foi encontrato por má evolução do processo evolutivo; um plano não foi encontrato pelo fato do mesmo não existir na instância de rede de Petri considerada. Sendo assim, após obter uma resposta negativa (*falso*) do método solve, pode-se optar por continuar o processo de expansão da rede considerada e posteriormente executar novamente o método *solve*. Com a finalidade de avaliar o desempenho do algoritmo proposto, as seções 4.3 e 4.4 consideraram apenas instâncias de redes de Petri nas quais existe pelo menos um plano.

Uma vez apresentado como os conceitos implementados até aqui foram agrupados para conceber o sistema planejador *GAPNet*, as próximas seções são reservadas para a

exposição dos resultados obtidos pelo mesmo.

4.3 Resultados preliminares

Os resultados obtidos com o sistema planejador *GAPNet* serão apresentados em duas seções: esta primeira, nomeada resultados preliminares, engloba as fases iniciais de testes e, portanto, não inclui em suas execuções algumas das otimizações apresentadas no capítulo 4; a próxima seção, resultados finais (4.4), apresenta o desempenho do planejador utilizando todas as funcionalidades implementadas.

Para analisar a implicação dos parâmetros utilizados sobre o processo evolutivo, serão plotados gráficos do comportamento da população a cada geração. Cada gráfico é composto por três curvas: curva da melhor aptidão, curva da aptidão média e curva da pior aptidão. Em evoluções bem sucedidas espera-se que a curva da aptidão média vá ao encontro da curva de melhor aptidão, determinando uma convergência da população e conseqüente resolução do problema.

Pela necessidade de exibir diferentes gráficos para um mesmo problema, foram anexadas identificações ("-00", "-01", etc) no final do nome de cada problema. Sendo assim o problema prob01-00 e prob01-01, por exemplo, representam execuções diferentes para um mesmo problema (prob01), possivelmente com uso de diferentes parâmetros. Vale ressaltar também que as tabelas de desempenho sempre apresentam execuções bem sucedidas, ou seja, que um plano foi encontrado. Quando uma execução mal sucedida estiver presente nessas tabelas, será incluído um identificador "não resolvido" ao lado do nome do problema. Ainda, os tempos apresentados nas tabelas de desempenho não consideram a fase de expansão das redes de Petri.

Os problemas considerados estão organizados de acordo com os domínios aos quais os mesmos pertencem. Enquanto a descrição em PDDL de cada problema é listada no anexo I, as descrições de domínios são apresentadas na própria seção referente ao mesmo. As considerações até então citadas são válidas, também, para a seção de resultados finais.

A fase inicial de testes foi realizada após o término das primeiras versões do GAPNet, contendo as seguintes características de implementação:

- a geração da população inicial é totalmente aleatória, conforme descrito pela seção 4.2.4 como sendo o método originalmente proposto;
- os operadores genéticos utilizados são os clássicos de mutação e de cruzamento;
- a função de aptidão utilizada foi o número de lugares satisfeitos na rede de Petri, conforme apresentado (seção 4.2.5);
- foi utilizado tanto o corte na rede (seção 4.2.7) quanto a otimização no produto matricial (seção 4.2.6);
- não foi testado o parâmetro *prioridade dentro de camadas* (seção 4.2.16);
- não foi utilizado o método que desconsidera as ações de manutenção (seção 4.2.8);
- não foram utilizados o novos operadores genéticos (seção 4.2.15);
- não foi utilizado o método de monitoramento (seção 4.2.14);

Os domínios analisados nesta fase de testes foram o mundo dos blocos e *gripper*.

4.3.1 Mundo dos blocos

O domínio dos blocos se caracteriza por reorganizar um conjunto de blocos dispostos em uma mesa a partir do uso de ações como empilhar e desempilhar bloco. Na seqüência, o arquivo de descrição em PDDL deste domínio:

```
(define (domain BLOCKS)
(:requirements :strips :typing)
(:types block)
(:predicates (on ?x - block ?y - block)
(ontable ?x - block)
        (clear ?x - block)
)
```

```
(:action stack
:parameters (?x - block ?y - block)
:precondition (and (clear ?x) (ontable ?x) (clear ?y) )
:effect
(and  (not (clear ?y))
(not (ontable ?x))
(on ?x ?y)))

(:action unstack
:parameters (?x - block ?y - block)
:precondition (and (on ?x ?y) (clear ?x))
:effect
(and   (clear ?y)
(ontable ?x)
(not (on ?x ?y)))))
```

Enquanto a tabela 4.1 mostra o desempenho do planejador *GAPNet* para algumas instâncias deste domínio, as imagens 4.1 a 4.8 mostram alguns gráficos de evolução da população para os mesmos. Vale salientar que, nesta tabela, apenas a última execução não encontrou um plano, destacada pelo indentificador "não-resolvido". A partir de agora, será feita uma análise dos gráficos gerados de acordo com a parametrização utilizada.

O primeiro problema a ser considerado é conhecido por anomalia de Sussman [35]. Tal problema (descrito no anexo I), apesar de parecer simples, pode representar grande dificuldade para muitos planejadores. Os gráficos das figuras 4.1 e 4.2 mostram duas evoluções obtidas para este problema.

Nestas duas execuções os seguintes parâmetros foram utilizados: TamanhoPopulacao: 50, TaxaCruzamento: 90%, TaxaMutacao: 30%, TaxaDiversos: 0, TamanhoTorneio: 4%, TipoAptidao: 1, TipoCruzamento: 1, TipoMutacao: 2, Profundidade: 2, PropagacaoPrioridades: 1

Planejador	**Problema**	**Tempo(s)**
gapnet	probBLOCKS-3-sussman-00	1.22
gapnet	probBLOCKS-3-sussman-01	0.67
gapnet	probBLOCKS-3-sussman-02	0.29
gapnet	probBLOCKS-3-sussman-03	0.34
gapnet	probBLOCKS-3-sussman-04	0.55
gapnet	probBLOCKS-3-sussman-05	0.39
gapnet	probBLOCKS-3-sussman-06	0.46
gapnet	prob01-00	0.12
gapnet	prob01-01	0.1
gapnet	prob02-00	1.75
gapnet	prob02-01	1.48
gapnet	prob02-02	1.39
gapnet	prob02-03	1.08
gapnet	prob02-04	1.7
gapnet	prob03-00	18.55
gapnet	prob03-01	27.77
gapnet	prob03-02	27.08
gapnet	prob04-00	146.4
gapnet	prob04-não-resolvido-00	145.19

Tabela 4.1: Resultados no domínio de blocos

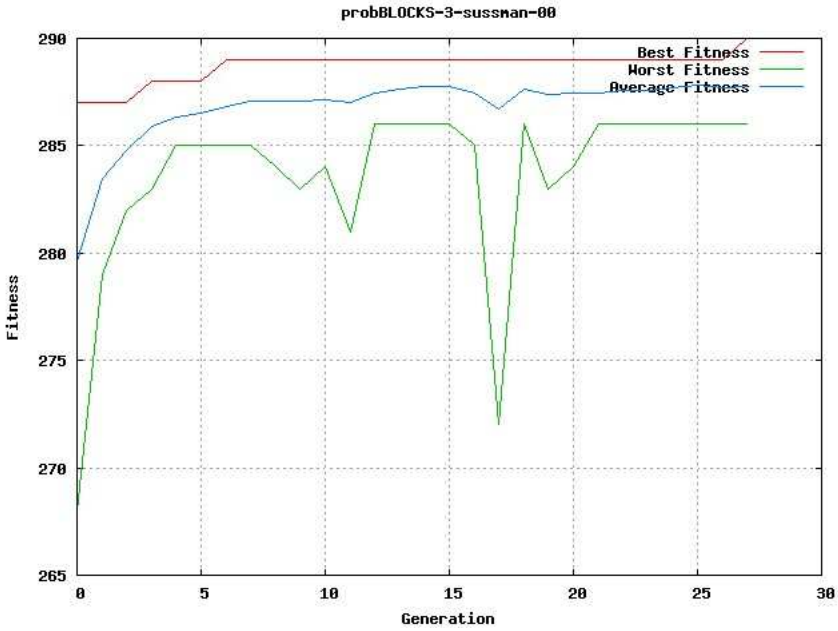

Figura 4.1: Gráfico de evolução para probBLOCKS-3-Sussman-00 (mundo dos blocos)

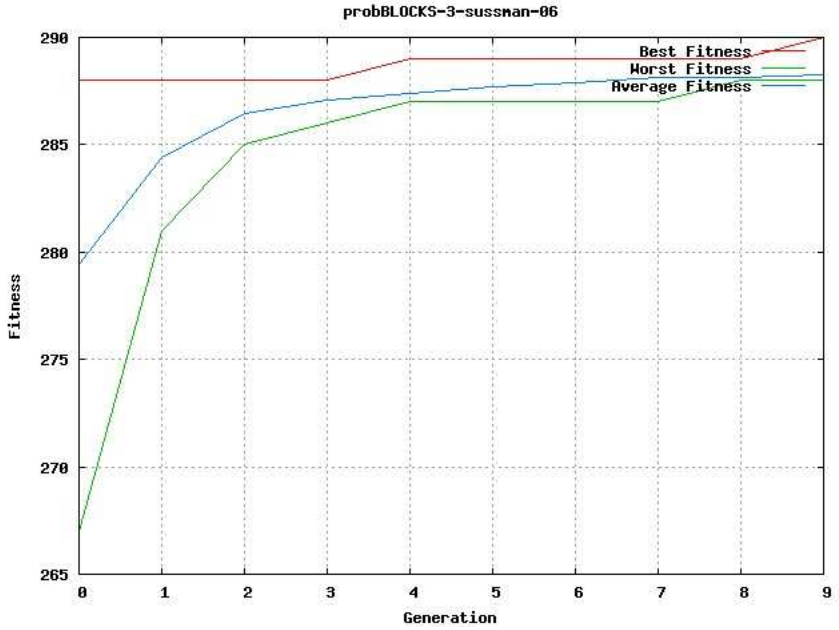

Figura 4.2: Gráfico de evolução para probBLOCKS-3-Sussman-06 (mundo dos blocos)

A diferença entre cada uma delas está no uso do elitismo e da população anterior. Enquanto a primeira utilizou um elitismo de 20%, a segunda considerou a população antiga na sua totalidade para atualização da população (parâmetro UsarPopulacaoAntiga). Assim como explicado na seção 4.2.13, o primeiro caso pode implicar na geração de novos indivíduos, justificando as bruscas variações encontradas na curva de pior aptidão do gráfico 4.1.

Os problemas prob01, prob02, prob03 e prob04 (descritos no anexo I) são pilhas formadas por 4, 5, 6 e 7 blocos respectivamente. O objetivo em cada um deles é desempilhar todos os blocos sobre a mesa.

Mais uma vez, o grande contraste nos gráficos gerados está relacionado ao uso do parâmetro UsarPopulacaoAntiga, o qual determina a forma de atualização da população, e a taxa de elitismo. Pode-se perceber, por exemplo, que no gráfico da figura 4.5 há um momento em que existe uma variação brusca no indivíduos da população e, logo após, o planejador encontra uma solução.

Essa variação, causada pela geração de novos indivíduos para completar a população, parece ser importante para redirecionar a busca feita pelo planejador. Tal característica poderia ser responsável por evitar máximos locais.

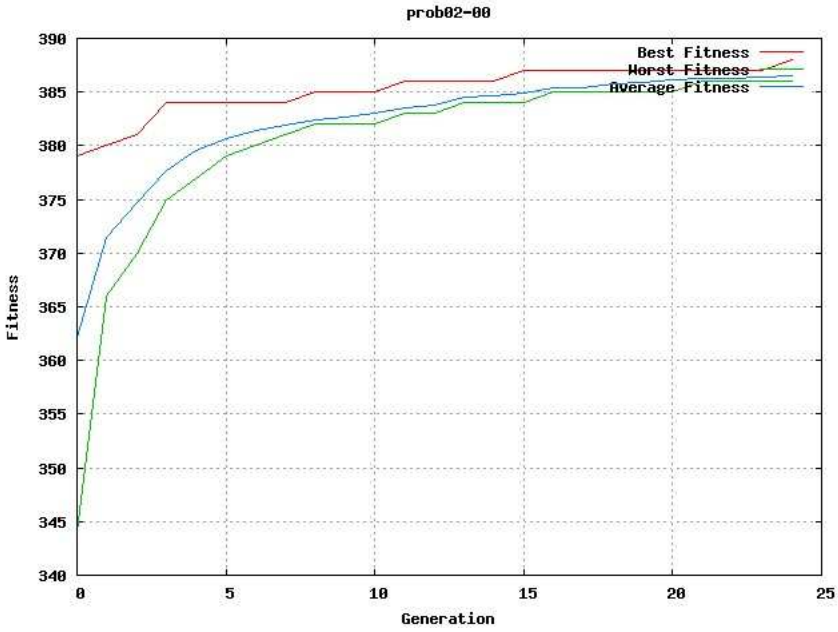

Figura 4.3: Gráfico de evolução para prob02-00 (mundo dos blocos)

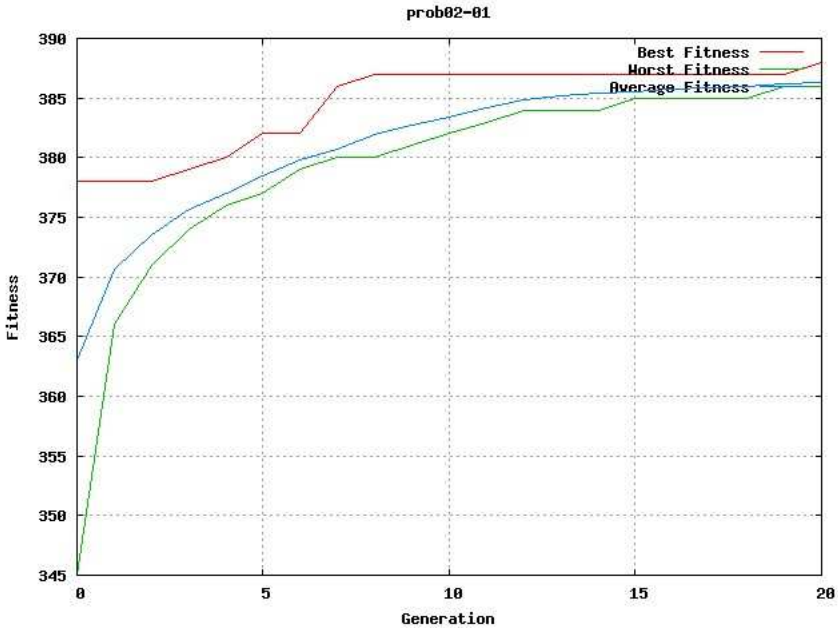

Figura 4.4: Gráfico de evolução para prob02-01 (mundo dos blocos)

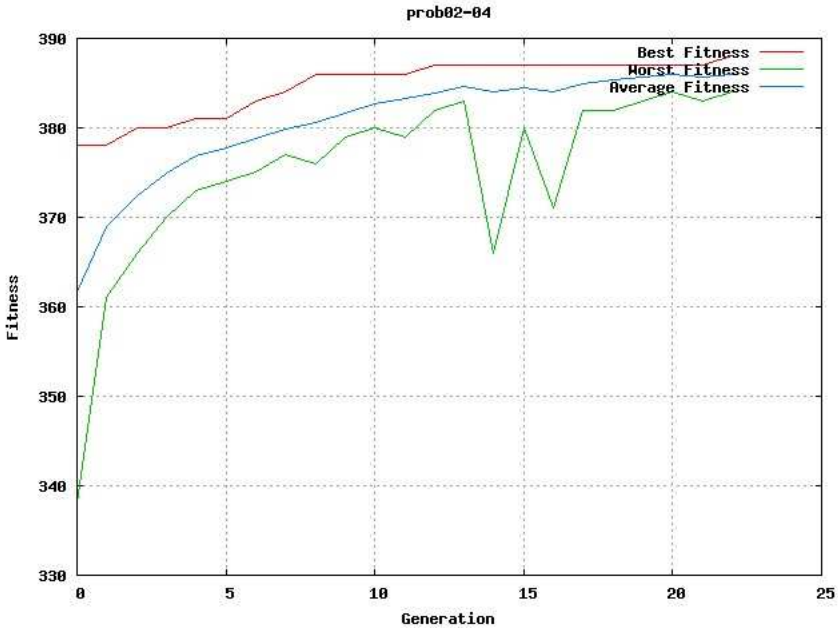

Figura 4.5: Gráfico de evolução para prob02-04 (mundo dos blocos)

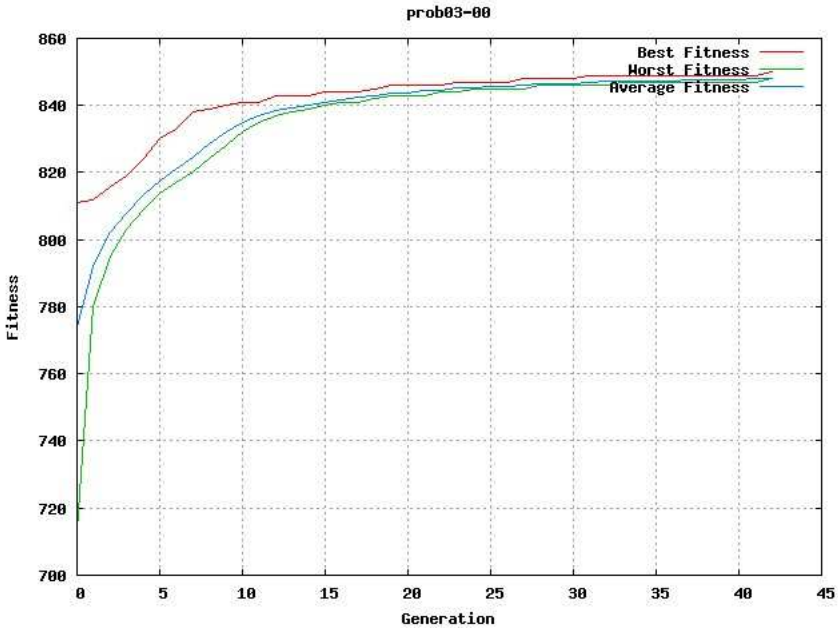

Figura 4.6: Gráfico de evolução para prob03-00 (mundo dos blocos)

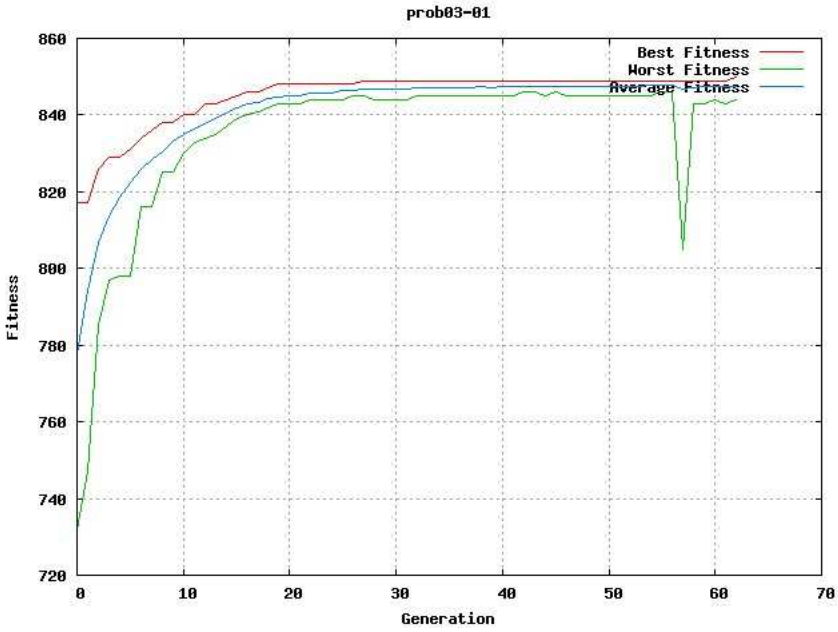

Figura 4.7: Gráfico de evolução para prob03-01 (mundo dos blocos)

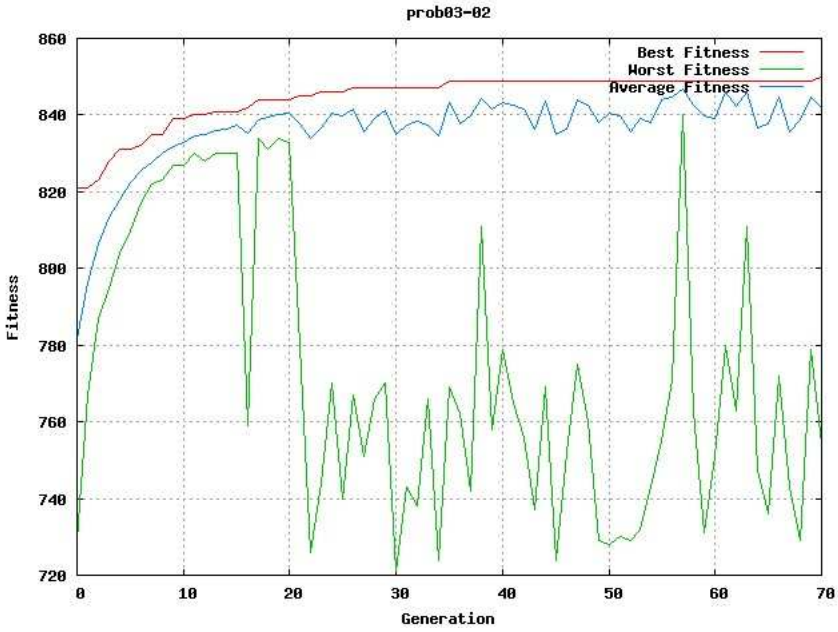

Figura 4.8: Gráfico de evolução para prob03-02 (mundo dos blocos)

Planejador	**Problema**	**Tempo(s)**
gapnet	prob00-00	1.51
gapnet	prob00-01	1.34
gapnet	prob00-02	2.30
gapnet	prob00-03	0.54
gapnet	prob00-04	1.33
gapnet	prob00-05	1.12
gapnet	prob00-06	0.84

Tabela 4.2: Resultados no domínio gripper

A partir dessas conclusões, optou-se também por modificar a atualização da população (ver seção 4.2.13), fazendo com que essas variações viessem a ocorrer com maior freqüência na população. Dessa forma, o parâmetro taxa de diversidade passou a ser responsável por manter uma população bem mais diversa ao longo do processo evolutivo.

Veja o gráfico da figura 4.8. Note que após algumas gerações a população começa a apresentar oscilações bem maiores de aptidão, permitindo que uma região maior do espaço de busca seja considerada.

Sendo assim, pôde-se notar três tipos diferentes de comportamento da população durante as evoluções obtidas para este domínio:

- A população da nova geração nunca apresenta uma aptidão menor do que a da geração anterior;
- Esporadicamente, a nova população apresenta uma aptidão menor do que a da geração anterior;
- A nova população apresenta, freqüentemente, grandes oscilações de aptidão com relação a da geração anterior.

4.3.2 *Gripper*

O domínio gripper, visto na seção 2.3, supõe a existência de algumas salas e um robô o qual deve movimentar objetos entre as mesmas. Para isso, o robô possui garras e também dispõe de ações como: pegar objetos, soltar objetos, e movimentar-se no ambiente.

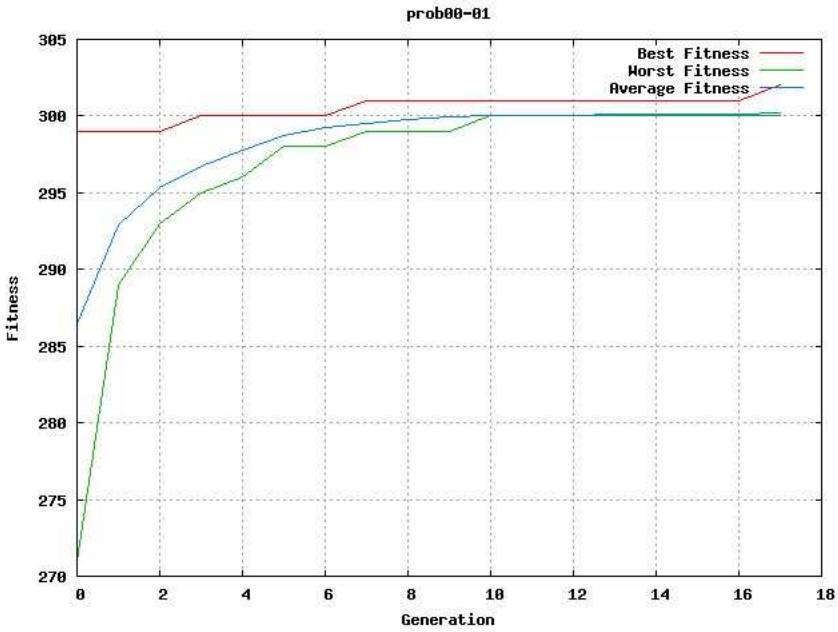

Figura 4.9: Gráfico de evolução para prob00-01 (*gripper*)

Assim como feito para o domínio anterior, é apresentada uma tabela com o desempenho do planejador *GAPNet* em cada execução. O problema prob00 (no anexo I) é constituído por duas bolas as quais devem ser movimentadas, por um robô, de uma sala para outra. Conforme pode ser visto na tabela 4.2, o algoritmo *GAPNet* foi capaz de solucionar este problema em todas execuções realizadas, usando as três formas de atualização de população (seção 4.2.13).

O gráfico da figura 4.11 mostra um caso em que a evolução da população ocorreu rapidamente até encontrar a solução. Já no gráfico da figura 4.10, percebe-se mais uma vez que a inserção de novos indivíduos na população contribui para a convergência final da mesma. Na figura 4.9 nota-se que mesmo usando o parâmetro UsarPopulacaoAntiga, o qual implica uma pressão evolutiva maior, um plano foi encontrado.

Embora o comportamento do algoritmo GAPNet tenha se mostrado adequado durante execuções para o problema prob00, seu desempenho para este domínio pode ser considerado ineficiente, visto a simplificade do único problema resolvido.

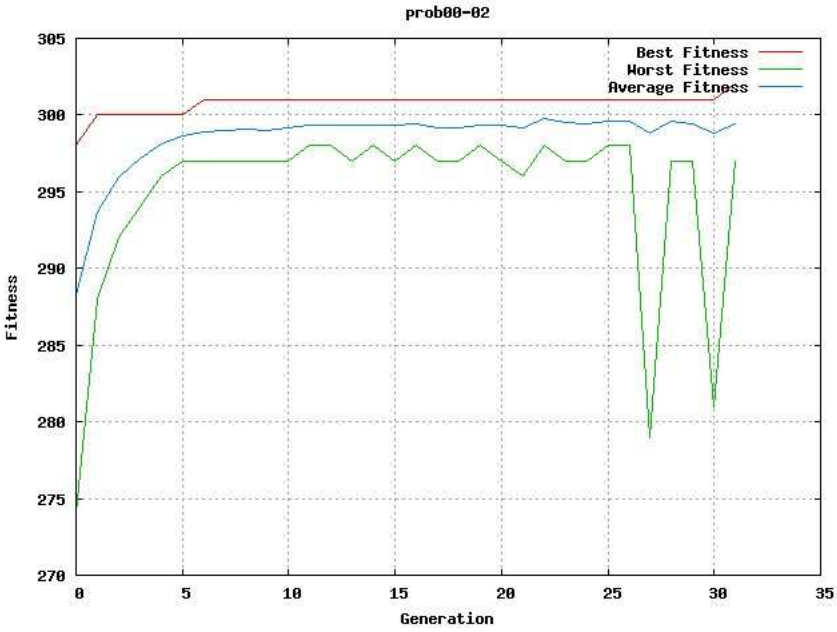

Figura 4.10: Gráfico de evolução para prob00-02 (*gripper*)

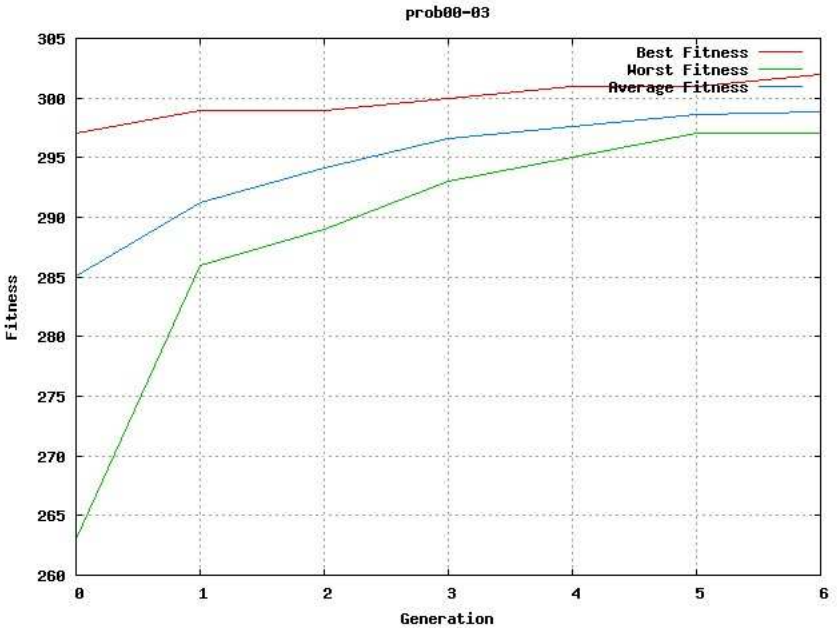

Figura 4.11: Gráfico de evolução para prob00-03 (*gripper*)

4.4 Resultados finais

Esta seção apresenta os resultados mais recentes obtidos pelo algoritmo *GAPNet*, mostrando como que as últimas funcionalidades implementadas melhoraram o seu desempenho. As considerações iniciais da seção 4.3 referente a organização dos problemas, seus nomes, seus identificadores e a plotagem dos gráficos, também valem para a presente seção. A única diferença é que os identificadore serão diferentes para os gráficos de evolução e para as tabelas de desempenho. Nas tabelas de desempenho serão anexados identificadores "a", "b", etc, para evitar confusão com os identificadores dos gráficos plotados ("00", "01", etc), visto que nem sempre o gráfico apresentado se refere ao tempo indicado na tabela. Os tempos aqui apresentados, assim como na seção anterior, desconsideram a fase de expansão da rede de Petri.

Os resultados insatisfatórios apresentados inicialmente foram responsáveis pela implementação de diversas otimizações e mudanças no algoritmo original. Os principais resultados obtidos, e listados a seguir, foram frutos de execuções as quais incorporaram as seguintes novas características:

- a geração da população inicial passou a ser mais variada quanto ao número de ações setadas como executáveis no cromossomo, assim como descreve a seção 4.2.4;
- as ações de manutenção passaram a ficar em segundo plano ("desconsideradas");
- a propagação na rede de Petri passou as ser executada conforme o algoritmo 8;
- a classificação de ações passou a considerar o parâmetro *prioridade dentro de camadas*, descrito pela seção 4.2.16;
- novos operadores genéticos de mutação e cruzamento, seção 4.2.15;
- uso de monitoramento;
- variações da função de aptidão, seção 4.2.17.

Conforme as novas características foram sendo adicionadas aos testes, obteve-se uma melhora significativa de desempenho no domínio *mundo dos blocos*, por outro lado, o

Planejador	**Problema**	**Tempo(s)**
gapnet	probBLOCKS-3-sussman-a	0.12
gapnet	probBLOCKS-3-sussman-b	0.12
gapnet	prob01-a	não considerado
gapnet	prob02-a	0.21
gapnet	prob02-b	0.23
gapnet	prob03-a	0.92
gapnet	prob03-b	1.83
gapnet	prob04-a	8.91
gapnet	prob04-b	8.92

Tabela 4.3: Resultados finais no domínio de blocos - grupo 1

domínio *gripper* mostrou-se novamente "difícil" para o planejador proposto. Dessa forma, optou-se por explorar novos domínios, como por exemplo *logistics* e *mystery*. Esse último, embora também na versão STRIPS, inclui características bem interessantes de capacidade e uso de combustível na locomoção de objetos.

Dentre os domínios tratados, o mundo dos blocos recebe destaque pelo fato de ter sido usado como referência na fase de ajustes e otimizações do sistema planejador. Os demais, *logistics* e *mystery*, foram considerados apenas para avaliar o desempenho do *GAPNet* em outros domínios, não tendo sido levados em consideração para refinar o sistema planejador. Os resultados obtidos em cada caso são apresentados nas próximas três seções.

4.4.1 Mundo dos blocos

Dentre os domínios analisados o mundo dos blocos foi, certamente, onde o sistema planejador *GAPNet* obteve seu melhor desempenho. A evolução com relação aos resultados obtidos anteriormente fica evidenciada, justificando algumas das otimizações propostas e implementadas.

Enquanto a tabela 4.3 apresenta os novos tempos de solução para os problemas considerados na seção 4.3, a tabela 4.4 lista uma série de novos problemas também considerados. Em ambas tabelas, todos os problemas foram solucionados. Essas informações permitem concluir que as otimizações propostas no algoritmo foram responsáveis por melhoras sensíveis em todos os problemas, além de permitir tratar e resolver outros de dificuldade maior.

Planejador	**Problema**	**Tempo(s)**
gapnet	prob05-a	35.65
gapnet	prob05-b	42.76
gapnet	probinverte05-a	12.12
gapnet	probinverte05-b	12.31
gapnet	probinverte06-a	60.25
gapnet	probinverte06-b	79.97
gapnet	probinverte07-a	206.54
gapnet	probinverte07-b	337.64
gapnet	probinverte08-a	589.2
gapnet	probinverte08-b	1356
gapnet	probinverte09-a	11035
gapnet	probinverte09-b	16132.4
gapnet	probinverte10-a	38082.2
gapnet	probinverte10-b	77246
gapnet	probinverte11-a	114351
gapnet	probinverte11-b	258887

Tabela 4.4: Resultados finais no domínio de blocos - grupo 2

Analisando inicialmente a tabela 4.3, percebe-se que para todos os problemas o algoritmo *GAPNet* melhorou seu desempenho. O problema *prob04* que, ora não era resolvido ora era resolvido com certa dificuldade, passou a ser facilmente tratado pelo planejador. Nessas execuções notou-se também que o fato de utilizar as novas características implementadas, conforme citadas anteriormente, foi de suma importância para o sucesso obtido. Pode-se dizer também que a variação de alguns parâmetros genéticos como taxa de mutação, cruzamento e elitismo, nestes casos, não apresentou um peso significativo nos tempos retornados.

O progresso obtido pelo planejador motivou a aumentar a dificuldade dos problemas considerados, assim como pode ser observado na tabela 4.4. O primeiro problema proposto nessa tabela de execuções, *prob05*, é um caso similar aos da tabela anterior, porém com um bloco a mais. Já os demais problemas, referentes a inversão de pilhas de blocos, envolvem explosões combinatoriais maiores.

Embora essa tabela mostre execuções bem sucedidas para inversão de até 11 blocos, os testes realizados indicam que o planejador começou a apresentar certa instabilidade a partir do problema *probinverte09*, o que implicou na não consideração de problemas maiores. Vale ressaltar também que foi imposto um limite de gerações, geralmente 1000,

Planejador	Problema	Execuções	Resolvidos	Percentual
gapnet	probBLOCKS-3-sussman-01	52	52	100%
gapnet	prob02	88	88	100%
gapnet	prob03	89	89	100%
gapnet	prob04	88	88	100%
gapnet	prob05	88	87	98.9%
gapnet	probinverte05	88	85	96.6%
gapnet	probinverte06	70	66	94.3%
gapnet	probinverte07	57	46	80.7%
gapnet	probinverte08	47	30	63.8%
gapnet	probinverte09	36	18	50%
gapnet	probinverte10	15	4	26.7%
gapnet	probinverte11	17	4	23.5%

Tabela 4.5: Resultados finais no domínio de blocos - percentual de acerto

para tratar esses problemas.

Mesmo que as tabelas 4.3 e 4.4 apresentem os melhores tempos obtidos pelo planejador *GAPNet* ao solucionar cada problema, sabe-se, por outro lado, que um algoritmo genético gera resultados diferentes inclusive quando executado com uma mesma parametrização. Dada essa característica, a tabela 4.5 faz uma análise de uma bateria de testes expondo o percentual de execuções bem sucedidas, buscando melhor qualificar o desempenho obtido. Essa tabela considera um grupo variado de parametrizações e, portanto, existem alguns parâmetros que individualmente apresentaram maior percentual de sucesso.

Além de analisar os tempos de execução e a taxa de acerto durante a solução de problemas, é importante também considerar de que forma se deu a evolução da população. Com essa finalidade, as figuras 4.12 a 4.19 apresentam alguns gráficos de comportamento da população durante a obtenção de soluções para os mesmos problemas. As execuções plotadas não necessariamente correspondem as mesmas execuções listadas na tabela 4.3.

A primeira característica a ser mencionada sobre estes gráficos diz respeito a nova abordagem utilizada para geração da população inicial. Observando as imagens 4.12 e 4.13 geradas para o problema prob02, e também as 4.14 e 4.15 geradas para o problema prob03, percebe-se que em todos os casos a população gerada apresenta uma aptidão média superior do que a gerada no testes preliminares. Essa situação pôde ser constatada em todos os problemas que foram reanalisados utilizando a mesma função de aptidão,

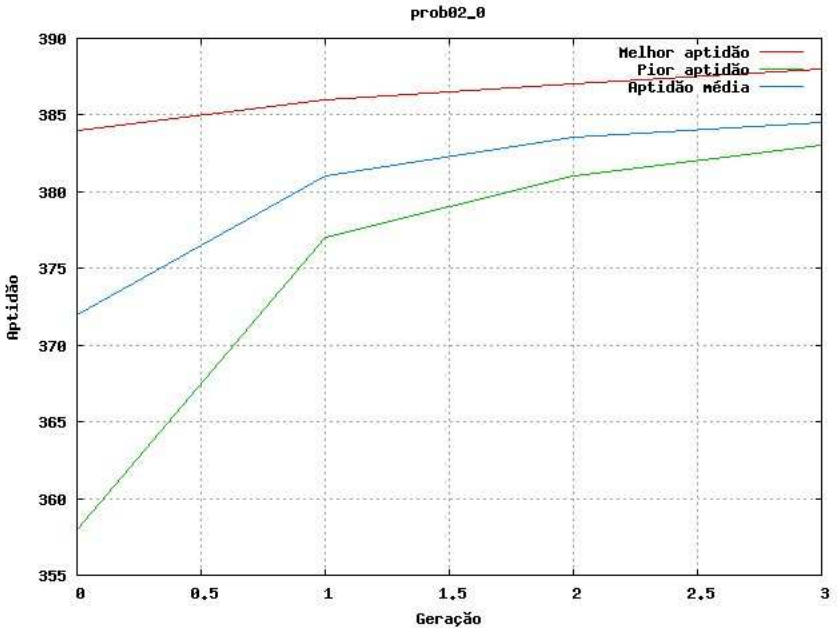

Figura 4.12: Gráfico de evolução para prob02-0 (mundo dos blocos)

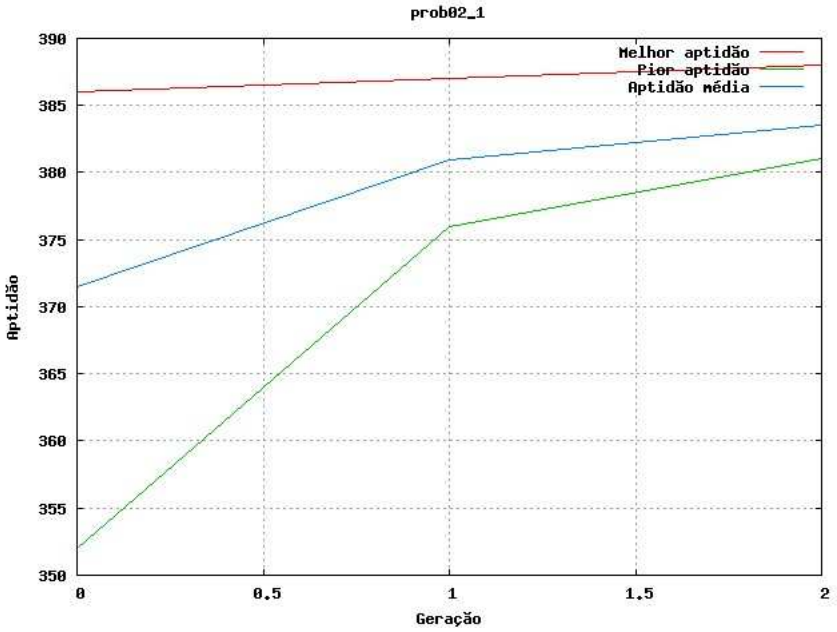

Figura 4.13: Gráfico de evolução para prob02-1 (mundo dos blocos)

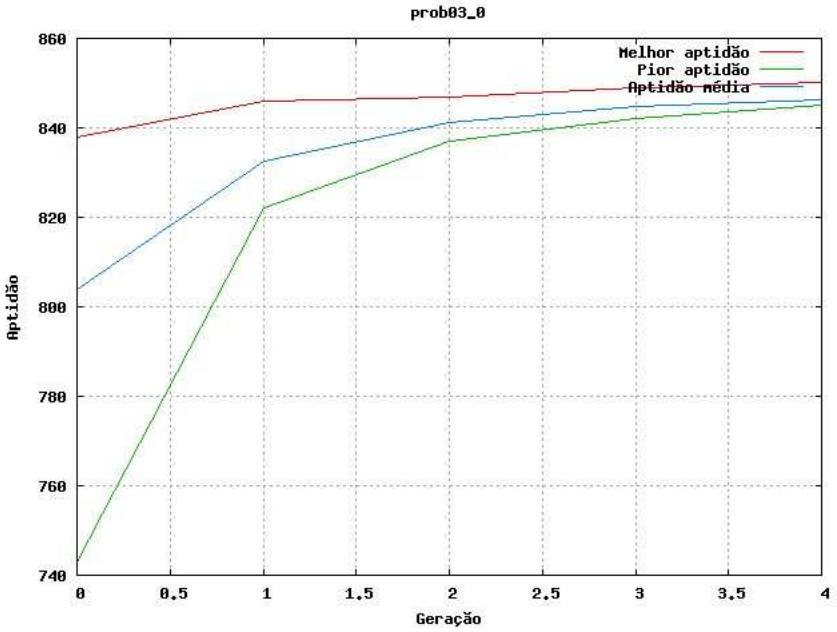

Figura 4.14: Gráfico de evolução para prob03-0 (mundo dos blocos)

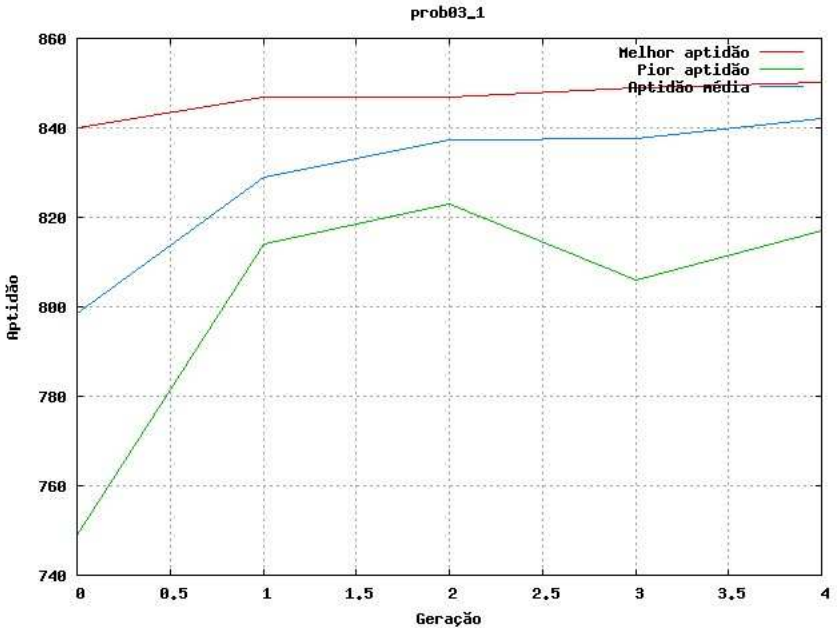

Figura 4.15: Gráfico de evolução para prob03-1 (mundo dos blocos)

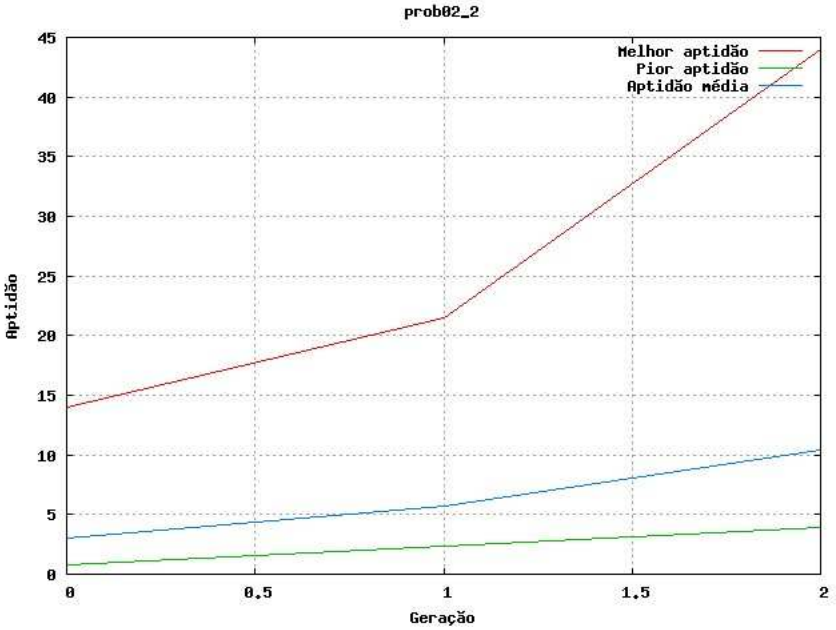

Figura 4.16: Gráfico de evolução para prob02-2 (mundo dos blocos)

neste caso, a função 1 (seção 4.2.5). O tamanho da população utilizada nestes casos foi de 100 indivíduos para as três primeiras execuções e 200 para a última.

Ainda quanto as imagens citadas anteriormente, pode-se notar que houve uma diminuição significativa no número de gerações necessárias para obtenção de uma solução. Essa melhora de desempenho é atribuída tanto ao novo método de geração da população inicial quanto a "desconsideração" das ações de manutenção (seção 4.2.8) e ao operador de reorganização de ações (seção 4.2.15.2). Dos gráficos plotados, somente o da imagem 4.15 pertence a uma execução na qual foi utilizado o novo operador de mutação baseado em conflitos.

Os gráficos das figuras 4.16 a 4.19, também ilustram execuções para os problemas prob02 e prob03, no entanto, nestes casos, foi utilizada uma função de aptidão alternativa definida como função 2 (seção 4.2.17). Quando a população, a execução 4.18 utilizou 300 indivíduos e as demais 100.

A partir deste momento serão analisados os novos problemas mencionados na tabela 4.5. As figuras 4.20 a 4.26 contém os gráficos de comportamento da população durante a resolução de alguns casos de inversão de pilha de blocos. Os dois primeiros, 4.20 e 4.21, mostram uma execução do algoritmo para o problema probinverte05, utili-

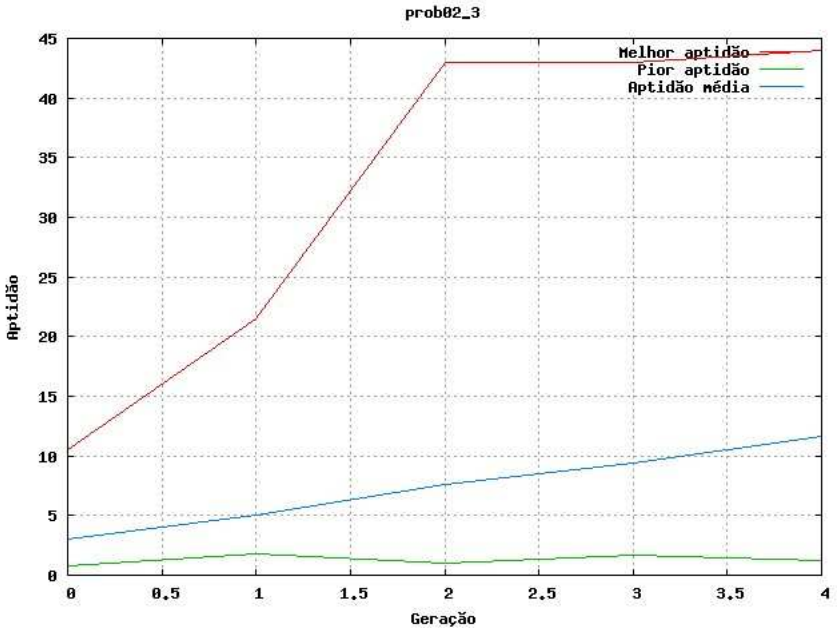

Figura 4.17: Gráfico de evolução para prob02-3 (mundo dos blocos)

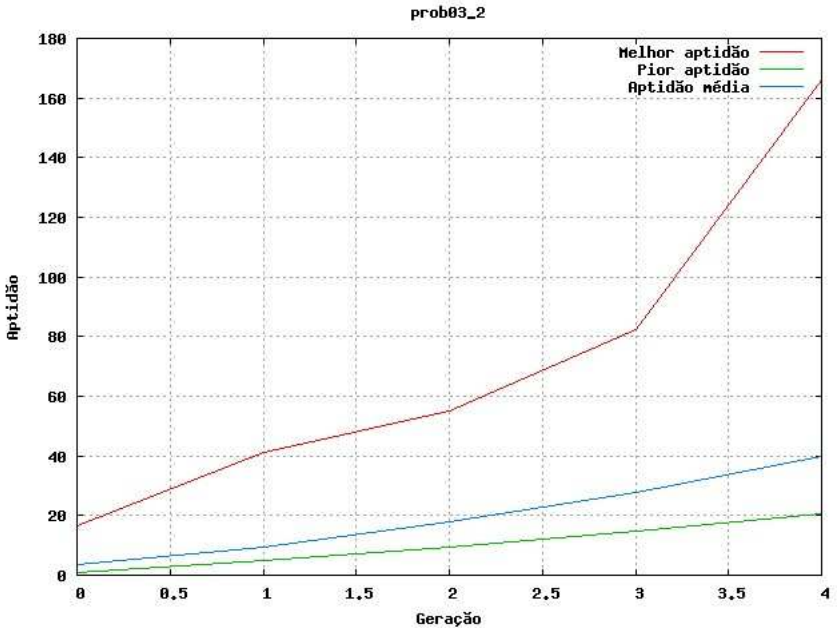

Figura 4.18: Gráfico de evolução para prob03-2 (mundo dos blocos)

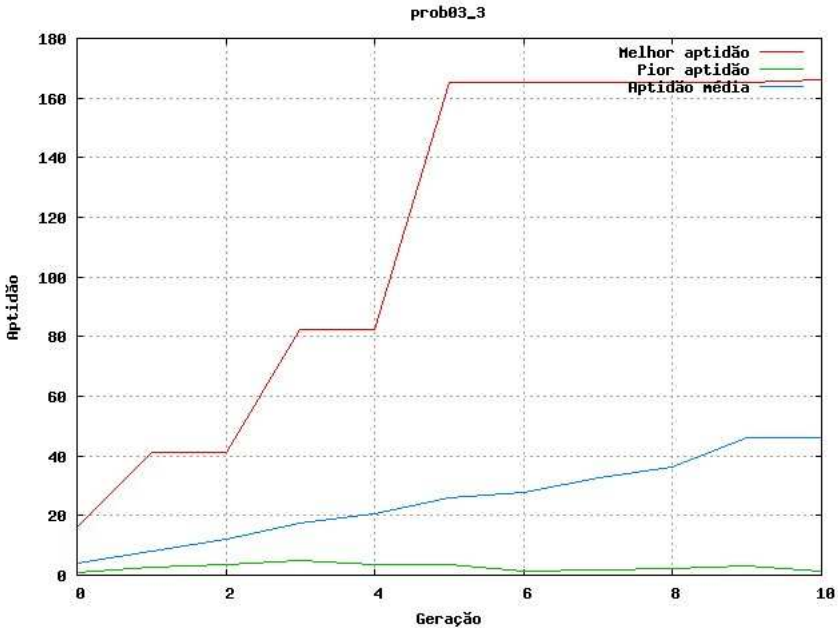

Figura 4.19: Gráfico de evolução para prob03-3 (mundo dos blocos)

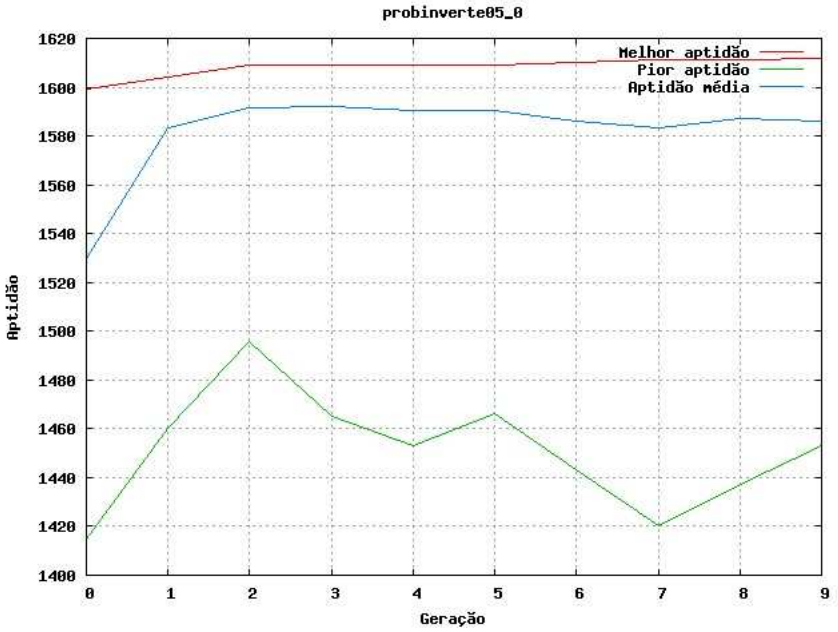

Figura 4.20: Gráfico de evolução para probinverte05-0 (mundo dos blocos)

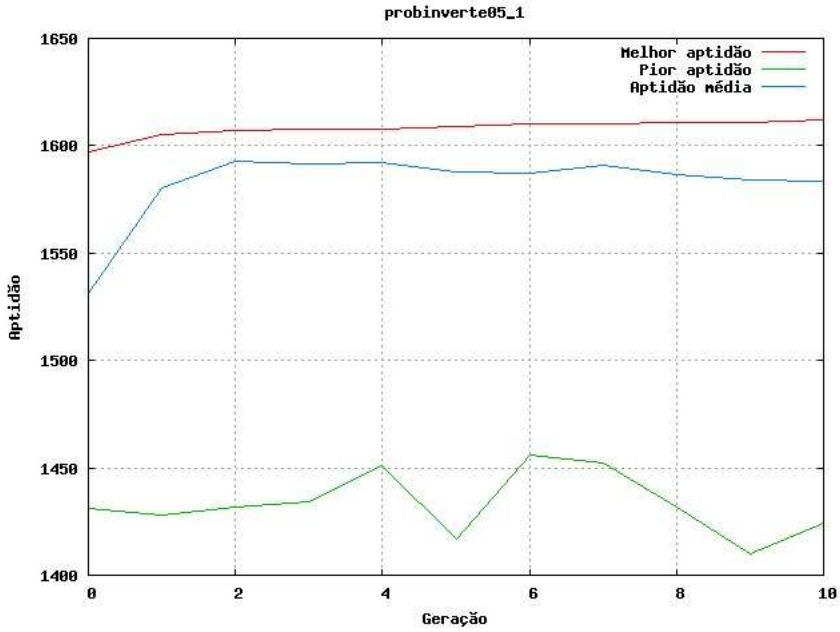

Figura 4.21: Gráfico de evolução para probinverte05-1 (mundo dos blocos)

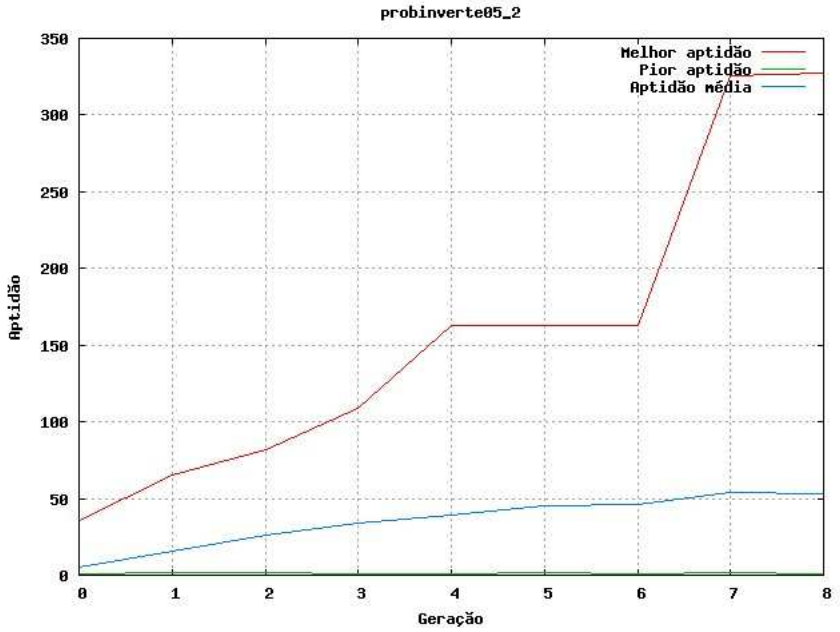

Figura 4.22: Gráfico de evolução para probinverte05-2 (mundo dos blocos)

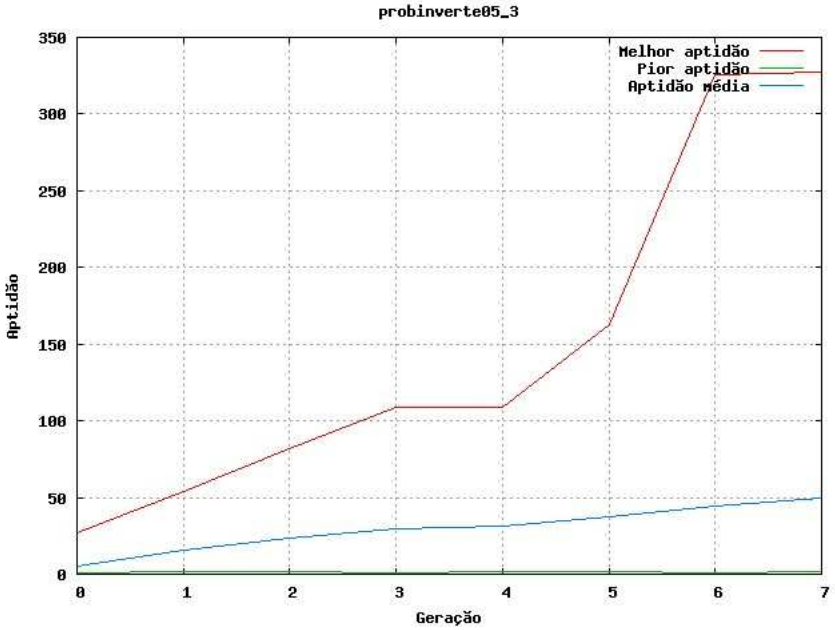

Figura 4.23: Gráfico de evolução para probinverte05-3 (mundo dos blocos)

zando a função de aptidão original (1) e variando apenas o tipo de mutação. Já os gráficos (4.22 e 4.23), são referentes à execuções utilizando a função de aptidão 2. Dentre esses quatro gráficos plotados para o problema probinverte05, a mutação baseada em conflitos foi utilizada no 4.21 e no 4.23.

Os últimos três gráficos (4.24, 4.25 e 4.26) também referentes ao problema de inversão de 5 blocos, ilustram o comportamento da população ao utilizar a função de aptidão número 3. O primeiro deles utilizou cruzamento 2 e mutação 1, o segundo usou cruzamento 1 e mutação 2, e, o terceiro, utilizou cruzamento 2 e mutação 2. Todas as execuções plotadas para o problema probinverte05 utilizaram população de 300 indivíduos.

NA maioria dos testes realizados foi utilizado uma taxa de cruzamento entre 65% e 80%. Para a mutação, foi utilizada uma taxa entre 10% e 20% quanto consideradas as a mutações 2 e a baseada em conflitos. Já para o caso da mutação tipo 1, foi usada uma taxa em torno de 5%. Na seleção por torneio, o tamanho do mesmo variou entre 5% e 15%.

Para o tratamento dos problemas maiores, foi sempre utilizada a atualização da população considerando uma taxa de elitistmo, sem utilizar na totalidade a população anterior. A classificação de ações mostrou-se mais adequada quando utilizou parâmetro de

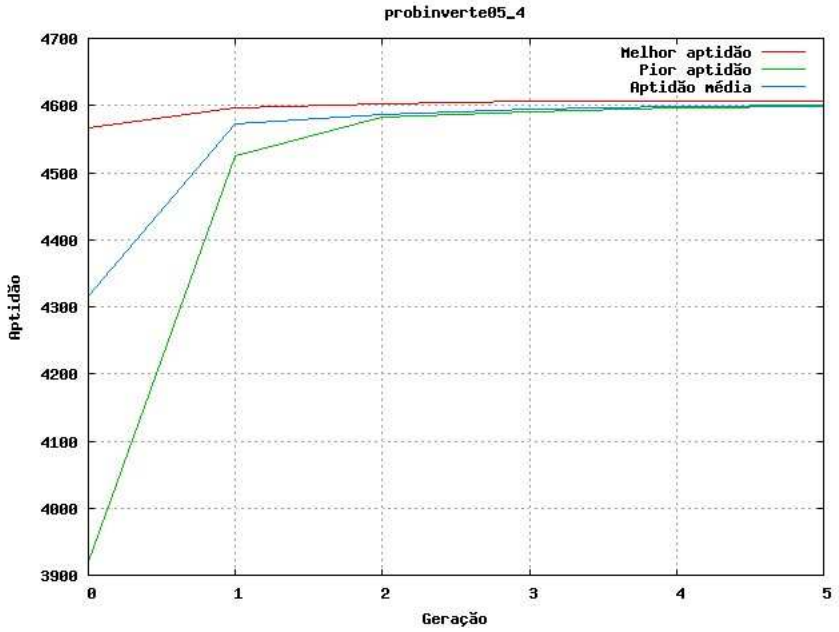

Figura 4.24: Gráfico de evolução para probinverte05-4 (mundo dos blocos)

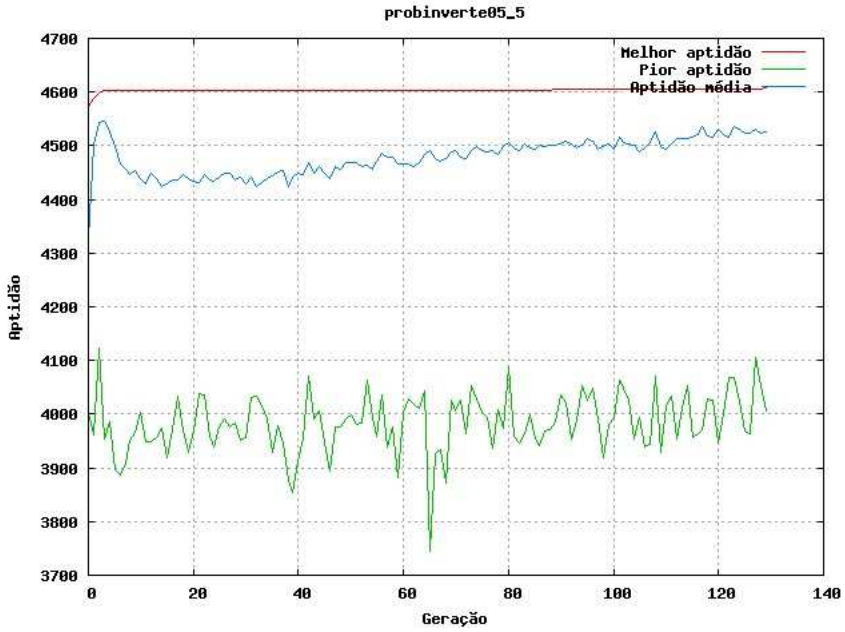

Figura 4.25: Gráfico de evolução para probinverte05-5 (mundo dos blocos)

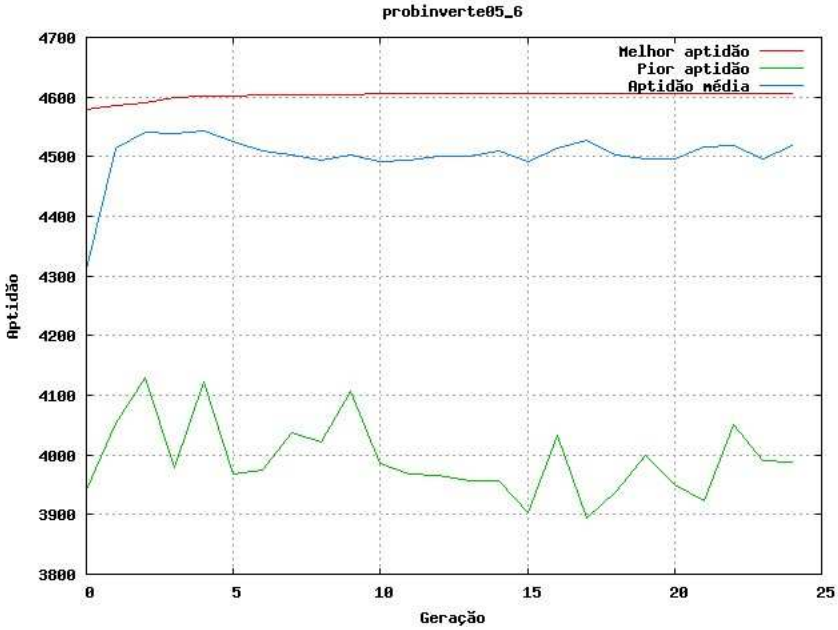

Figura 4.26: Gráfico de evolução para probinverte05-6 (mundo dos blocos)

profundidade igual a um (1) e, ao mesmo tempo, considerou as prioridades dentro de camadas (4.2.16 ao invés da propagação de influências. O operador de reorganização de ações foi utilizado, na maioria dos teste, com uma probabilidade de execução igual a 50%.

Também com o intuito de melhor qualificar as análises, a tabela 4.6 apresenta um comparativo de desempenho entre alguns planejadores (tempo em segundos). Além do *GAPNet*, também são inclusos o *AGPlan* (seção 4.1.2), o *Graphplan*, *Blackbox* e o *FF-Plan*. Dentre estes, somente o *Blackbox* é implementado fora do IPE. *FFPlan* é uma implementação simplificada do *FF*, sem considerar a busca local. É importante destacar que os tempos indicados para o *GAPNet* são geralmente os melhores tempos obtidos pelo mesmo em uma bateria de execuções. Para casos esporádicos onde um tempo muito baixo foi obtido, o mesmo não foi considerado. Sendo assim, casos isolados de sucesso não estão presentes nesta tabela.

Quanto aos tempos gerados pelo planejador *AGPlan*, foi selecionado um dos melhores tempos obtidos em uma série de execuções. Ao contrário do *GAPNet*, entretanto, o *AGPlan* mostrou-se bem mais estável, sendo capaz de solucionar os problemas em praticamente todas execuções realizadas e ao mesmo tempo sem exigir muita variação dos parâmetros genéticos.

Problema	***GANet***	***AGPlan***	***Graphplan***	***Blackbox***	***FFPlan***
probBLOCKS-3-sussman	0.12	0.02	0.02	0	0.01
prob02	0.21	0.01	0.03	0	0.02
prob03	0.92	0.17	0.07	0	0.05
prob04	8.91	0.2	0.17	0.01	0.08
prob05	35.65	0.3	0.42	0.01	0.14
probinverte05	12.31	0.12	0.17	0.01	0.05
probinverte06	60.25	0.12	0.36	0.01	0.09
probinverte07	206.54	1.5	0.8	0.02	0.16
probinverte08	589.2	1.8	1.5	0.03	0.26
probinverte09	11035	1.7	2.84	0.04	0.41
probinverte10	38082	2.0	4.75	0.06	0.65
probinverte11	114351	1.9	8.0	0.08	0.95

Tabela 4.6: Comparativo entre planejadores (tempo em segundos) - mundo dos blocos

Em muitas análises, pôde-se constatar que o número de gerações processadas pelo algoritmo *GAPNet* e pelo *AGPlan*, são muito semelhantes. Por outro lado, é significativa a diferença de tempo exigido em cada planejador para realizar a avaliação de seus indivíduos. Enquanto a validação de um cromossomo no *AGPlan* é feita de forma muito rápida, no *GAPNet* isso exige um série de cálculos matriciais, o que despende muito tempo de processamento. Por conseqüência, o aumento do tamanho da população no *GAPNet*, embora baixe o número de gerações em alguns casos, aumenta consideravelmente o tempo de processamento.

As duas próximas seções apresentam alguns resultados obtidos para o domínio *logistics* e *mystery*, respectivamente.

4.4.2 Logistics

O domínio *logistics* se caracteriza pelo transporte de objetos entre diferentes localidades, podendo utilizar para isso veículos como aviões e caminhões. Na versão testada neste trabalho, conforme declaração abaixo, é considerado um domínio contendo apenas caminhões.

```
(define (domain logistics-strips)
(:types   truck - vehicle
```

```
          package
          vehicle - physobj
          location - place
          place
          physobj - object)

(:predicates
(at ?obj - physobj ?loc - place)
(in ?pkg - package ?veh - vehicle))

(:action LOAD-TRUCK
   :parameters    (?pkg - package ?truck - truck ?loc - place)
   :precondition  (and (at ?truck ?loc) (at ?pkg ?loc))
   :effect        (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))

(:action UNLOAD-TRUCK
   :parameters   (?pkg - package ?truck - truck ?loc - place)
   :precondition (and (at ?truck ?loc) (in ?pkg ?truck))
   :effect       (and (not (in ?pkg ?truck)) (at ?pkg ?loc)))

(:action DRIVE-TRUCK
   :parameters (?truck - truck ?loc-from - place ?loc-to - place)
   :precondition (and (at ?truck ?loc-from) )
   :effect (and (not (at ?truck ?loc-from)) (at ?truck ?loc-to)))
)
```

A maioria dos problemas testados nesse domínio são considerados bem simples, envolvendo planos de poucas ações. A tabela 4.7 mostra os tempos obtidos pelo planejador *GAPNet* durante algumas execuções. Conforme pode ser observado nos problemas tratados, os últimos dois não foram solucionados, porém, até então, não foram realizados

Planejador	**Problema**	**Tempo(s)**
gapnet	prob001-a	0.05
gapnet	prob001-b	0.06
gapnet	prob002-a	0.04
gapnet	prob002-b	0.04
gapnet	prob003-a	0.63
gapnet	prob003-b	0.92
gapnet	prob004-a	27.45
gapnet	prob004-b	34.96
gapnet	prob005-a	0.12
gapnet	prob005-b	0.13
gapnet	prob006-a	38.37
gapnet	prob006-b	39.71
gapnet	prob007-não-resolvido	x
gapnet	prob008-não-resolvido	x

Tabela 4.7: Resultados no domínio logistics

Planejador	**Problema**	**Execuções**	**Resolvidos**	**Percentual**
gapnet	prob001	86	86	100%
gapnet	prob002	86	86	100%
gapnet	prob003	74	64	86.5%
gapnet	prob004	56	30	53.4%
gapnet	prob005	67	67	100%
gapnet	prob006	30	28	93.3%
gapnet	prob007	9	0	0.%
gapnet	prob008	9	0	0.%

Tabela 4.8: Resultados no domínio *logistics* - percentual de acerto

muitos testes neste domínio. A tabela 4.8 mostra a taxa de acerto para cada problema.

Vale ressaltar que este domínio não foi devidamente explorado, os resultados apresentados são os obtidos para execuções com parametrizações semelhantes às usadas no mundo de blocos, ou seja, não foi houve uma tentativa de ajuste de parâmetros para melhor solucionar este domínio. Por último, a tabela 4.9 apresenta os tempos (em segundos) obtido por cada planejador para o presente domínio. Mais uma vez, os tempos considerados para os planejadores *GAPNet* e *AGPlan* foram alguns dos melhores obtidos, desde que não tenham ocorrido isoladamente.

Problema	*GANet*	*AGPlan*	*Graphplan*	*Blackbox*	*FFPlan*
prob001	0.05	0	0	0	0.01
prob002	0.04	0	0	0	0.01
prob003	0.63	0	0.01	0	0.02
prob004	27.45	0.01	0.06	0	0.19
prob005	0.12	0	0.02	0	0.04
prob006	38.37	0.03	0.1	0	0.13
prob007	x	0.09	0.2	0	0.98
prob008	x	0.8	0.33	0	1.61

Tabela 4.9: Comparativos entre planejadores (tempo em segundos) - *logistics*

4.4.3 Mystery

Mystery é um domínio de transporte de objetos que inclui algumas características muito interessantes como: limitação da capacidade de espaço físico em meios de transporte; e também consumo de combustível durante a execução de deslocamentos. Para que tais características pudessem ser utilizadas no ambiente IPE, foi adotada uma versão STRIPS desse domínio. Uma descrição do mesmo é apresentada abaixo.

```
(define (domain mystery-strips)
   (:requirements :strips :typing)
   (:types space fuel
           location vehicle cargo)
   (:predicates
       (at ?v ?l)
       (conn ?l1 ?l2)
       (has-fuel ?l ?f)
       (fuel-neighbor ?f1 ?f2)
       (in ?c ?v)
       (has-space ?v ?s)
       (space-neighbor ?s1 ?s2)

   )
```

```
(:action move
    :parameters (?v - vehicle ?l1 ?l2 -location ?f1 ?f2 - fuel)
    :precondition (and (at ?v ?l1)
                       (conn ?l1 ?l2)
                       (has-fuel ?l1 ?f1)
                       (fuel-neighbor ?f2 ?f1))
    :effect (and (not (at ?v ?l1))
                 (at ?v ?l2)
                 (not (has-fuel ?l1 ?f1))
                 (has-fuel ?l1 ?f2)))

(:action load
    :parameters (?c - cargo ?v - vehicle ?l -location
                 ?s1 ?s2 - space)
    :precondition (and (at ?c ?l)
                       (at ?v ?l)
                       (has-space ?v ?s1)
                       (space-neighbor ?s2 ?s1))
    :effect (and (not (at ?c ?l))
                 (in ?c ?v)
                 (not (has-space ?v ?s1))
                 (has-space ?v ?s2)))

(:action unload
    :parameters (?c - cargo ?v - vehicle ?l - location
                 ?s1 ?s2 - space)
    :precondition (and (in ?c ?v)
                       (at ?v ?l)
```

Planejador	Problema	Tempo(s)
gapnet	prob00-a	0.12
gapnet	prob00-b	0.13
gapnet	prob01-a	24.09
gapnet	prob01-b	27.63
gapnet	prob02-não-resolvido	x

Tabela 4.10: Resultados no domínio mystery

Planejador	Problema	Execuções	Resolvidos	Percentual
gapnet	prob00	39	39	100%
gapnet	prob01	40	39	97.5%
gapnet	prob02	10	0	0.0%

Tabela 4.11: Resultados no domínio *mystery* - percentual de acerto

```
                    (has-space ?v ?s1)
                    (space-neighbor ?s1 ?s2))
    :effect (and (not (in ?c ?v))
                 (at ?c ?l)
                 (not (has-space ?v ?s1))
                 (has-space ?v ?s2)))
)
```

Os problemas tratados nesse domínio consideram condições mais realistas em um sistema de transporte, o que os tornam muito atraentes. Por outro lado, situações envolvendo poucos objetos e poucas localidades são suficientes para dificultar a busca por uma solução.

Enquanto a tabela 4.10 contém a relação de tempos obtidos para solução de cada problema no domínio mystery, onde somente o problema *prob02* não foi resolvido, a tabela 4.11 apresenta a taxa de acerto obtida em uma determinada bateria de testes.

Neste domínio, são plotados três gráficos (4.27, 4.28 e 4.29) de comportamento da população para execuções do problema prob01. Este problema, assim como os demais propostos neste domínio, implicaram certa dificuldade para outros planejadores, conforme será verificado na tabela 4.12

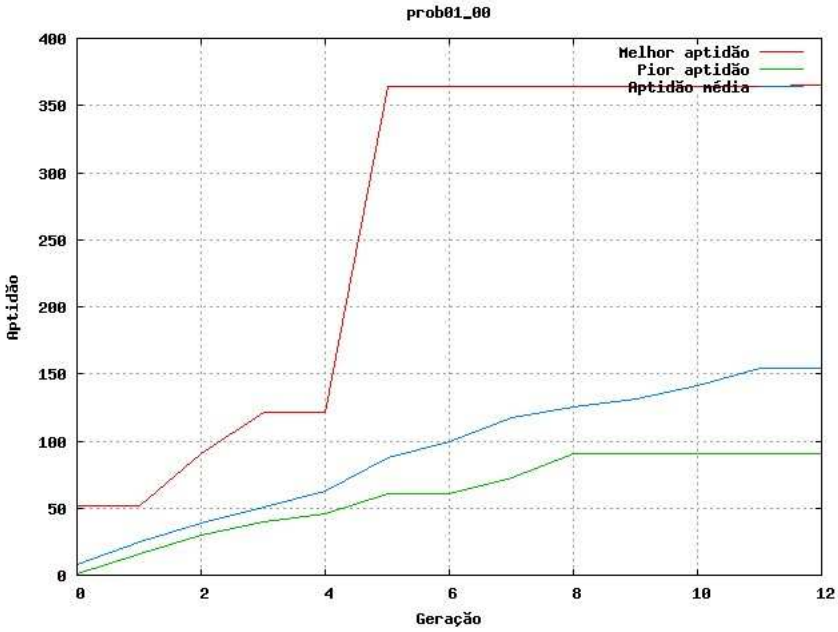

Figura 4.27: Gráfico de evolução para prob01-00 (*mystery*)

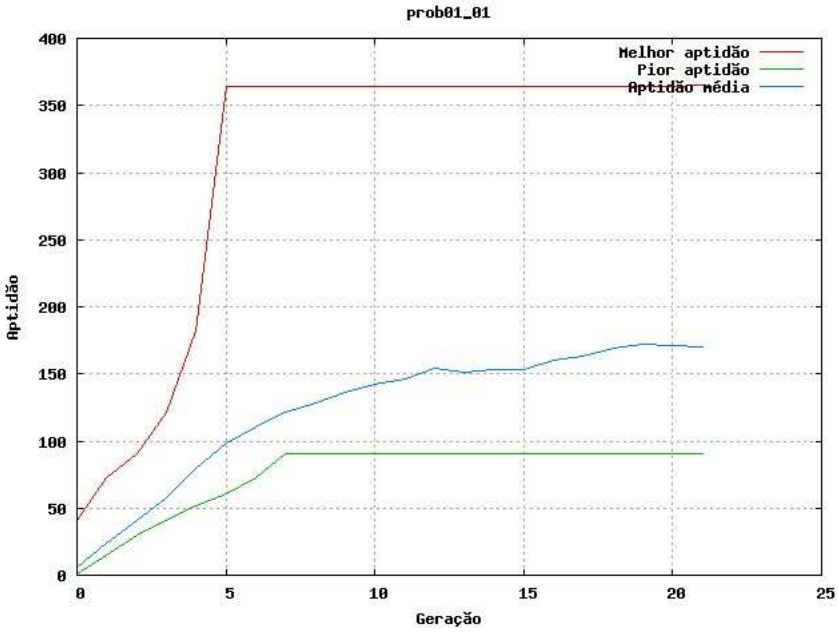

Figura 4.28: Gráfico de evolução para prob01-01 (*mystery*)

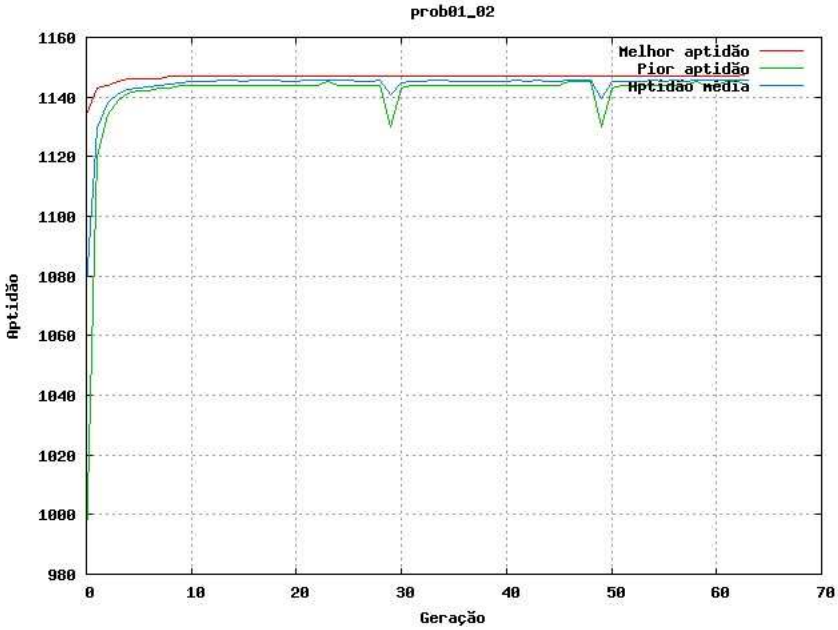

Figura 4.29: Gráfico de evolução para prob01-02 (*mystery*)

Problema	***GANet***	***AGPlan***	***Graphplan***	***Blackbox***	***FFPlan***
prob00	0.12	0.14	0.04	0	-
prob01	24.09	0.18	-	0.01	0.19
prob02	x	2.8	-	1.0	-

Tabela 4.12: Comparativos entre planejadores (tempo em segundos) - *mystery*

Os dois primeiros, 4.27 e 4.28, são a plotagem de duas execuções utilizando a função de aptidão número 2. O restande de seus parâmetros são praticamente idênticos, exceto pelo número de indivíduos. O primeiro caso tem uma população de tamanho 100, enquanto o segundo de 300.

O último gráfico plotado neste domínio foi escolhido com a intenção de mostrar como que, eventualmente, pode ocorrer a execução do monitoramento (seção 4.2.14). Na execução da figura 4.29, foi utilizada a função de aptidão 1, e também o monitoramento estava ativado para identificar caso a população ficasse 20 gerações sem alterar seu melhor indivíduo. Uma vez identificada esta situação, ocorre a inserção de novos indivíduos com intuito de diversificar o conjunto de soluções candidatas. Logo após a ocorrência de duas situações destas, um plano foi encontrado.

A tabela 4.12 mostra um comparativo (em segundos) dos tempos obtidos por cada planejador para o domínio *mystery*. Assim como nas comparações anteriores, foram con-

siderados para o *GAPNet* os melhores resultados, desde que os mesmos não tenham ocorrido isoladamente. O problema prob02, por exemplo, não foi solucionado pelo planejador proposto neste trabalho. Por outro lado, o planejador *AGPlan* mostrou-se mais uma vez eficiente, conseguindo solucionar todos os problemas considerados.

Curiosamente, neste domínio, o *FFPlan* e o *graphplan* (ambos versão do IPE) não retornaram uma resposta. Nas execuções realizadas e indicadas com "-" na tabela, seus algoritmos ficaram em processamento durante um tempo bem superior ao dos demais planejadores, quando tiveram sua execuções interrompidas.

Embora os problemas tratados neste domínio possam parecer simples, as novas características introduzidas no mesmo parecem dificultar a busca por soluções em alguns planejadores. É provável que futuros ajustes no *GAPNet* levem-no a obter resultados competitivos neste domínio.

CAPÍTULO 5

CONCLUSÃO

Este trabalho apresentou uma abordagem evolutiva com a finalidade de tratar problemas de alcançabilidade em uma determinada classe de redes de Petri acíclicas e, por conseqüência, resolver problemas de planejamento em inteligência artificial. O algoritmo proposto tem como princípio tratar um conjunto de transições conflitantes, buscando encontrar uma correta seqüência de disparos na rede em análise.

Para tal, foi feita uma revisão do estado da arte na área de planejamento, incluindo a representação STRIPS, o *Graphplan* e a linguagem de definição de domínios PDDL. Também foram apresentados o ambiente IPE e a representação baseada em redes de Petri, ambos desenvolvidos por pesquisadores do Departamento de Informática da UFPR. Foram analisadas também propostas de uso de algoritmos genéticos em planejamento utilizando a representação STRIPS, o grafo de planos e fórmulas SAT.

Foi visto que a classe *petrinet* do ambiente IPE realiza uma modelagem de problemas de planejamento utilizando as mesmas idéias do grafo de planos. Adicionalmente, o formalismo das redes de Petri permite um melhor entendimento dos conflitos existentes no problema em questão, além de fornecer um conjunto de equações baseadas em cálculo matricial o qual facilita a análise da dinâmica de sistemas.

O sistema planejador *GAPNet* propõe uma técnica evolucionária para resolver o problema de alcançabilidade nas redes de Petri geradas pela classe *gapnet*. As funções de aptidão utilizadas pelo algoritmo genético levam em consideração o número de lugares satisfeitos na rede e também o número de conflitos (*mutex's*) satisfeitos.

Os resultados obtidos pelo sistema planejador proposto foram separados em duas seções, e também organizados de acordo com os domínios considerados. Os testes preliminares analisaram principalmente o comportamento da população ao longo das gerações. Percebeu-se que para problemas menores, o desempenho do sistema mostrou-se estável

independente do tipo de atualização da população utilizado. Por outro lado, para problemas maiores, a atualização da população baseada no algoritmo 9 mostrou-se menos eficiente.

Para os testes finais, foram realizadas algumas otimizações as quais levaram a uma sensível melhora nos resultados. Tanto a modificação na geração da população inicial quanto a "desconsideração" das ações de manutenção mostraram-se importantes nesse ganho de desempenho.

Uma análise de um conjunto de soluções candidatas, com relação a classificação de ações, levou a algumas conclusões importantes. Esta classificação envolve parâmetros como profundidade da árvore de influências, propagação das influências e também o uso de prioridade dentro de camadas. Foi constatado que, para vários problemas, era mais vantajoso utilizar uma profundidade igual a um (1) e ao mesmo tempo considerar a prioridade dentro de camadas, ao invés de realizar uma propagação das influências.

Além da obtenção de alguns resultados, isso pôde ser observado da seguinte forma: selecionou-se alguns indivíduos e realizou-se uma mutação para cada um de seus genes. Para cada uma dessas mutações, foi realizada uma reavaliação a partir da função de aptidão. Quando usados os parâmetros mencionados a cima, percebeu-se que a alteração da aptidão tinha uma relação com a posição do gene mudado no cromossomo. Essa característica era a intenção da classificação de ações, ou seja, organizar o cromossomo de forma a diferenciar os genes de acordo com o peso ou influência que cada um tem sobre a rede representada. Isso levou a utilização desta configuração nos próximos testes.

Quanto a criação dos novos operadores genéticos, foi identificado que o operador de reorganização de ações garante, em alguns casos, uma melhor evolução da população. Por outro lado, o operador de mutação baseado em conflitos nem sempre garante uma correta evolução da população. O sucesso das soluções obtidas está bem mais relacionado ao tipo de atualização da população e a função de aptidão, do que a pequenas variações nas taxas dos operadores genéticos.

Na seção de resultados finais foram apresentadas algumas tabelas indicando uma taxa de acerto do planejador *GAPNet* para cada problema. Naquelas tabelas, foram utiliza-

das diversas parametrizações diferentes. Pode-se dizer que, quando a função de aptidão considerada é a de número 2 ou 3, a taxa de acerto é um pouco superior, significando que a função de aptidão 1 mostrou-se menos eficiente que as demais.

Embora para muitos problemas uma solução tenha sido encontrada com poucas gerações, o tempo de processamento obtido pelo planejador foi bem elevado em relação ao demais analisados. Numa comparação da presente proposta com a do algoritmo genético baseado no grafo de planos, observa-se que o número de gerações é, muitas vezes, equivalente. O gargalo existente no *GAPNet* são os sucessivos cálculos matriciais exigidos para propagação da rede de Petri durante a avaliação de cada solução candidata.

Alternativamente, poderia-se optar por outra forma de avaliação, a qual evitasse o cálculo matricial. Entretanto, manter a capacidade de tratar um problema de alcançabilidade nessas redes é muito interessante, visto que pode permitir solucionar uma variedade maior de problemas futuramente. Imagine que a classe *petrinet* venha a incorporar características de planejamento não clássico. Nesta situação, um planejador capaz de solucionar essas redes poderá ser adaptado para expandir o seu domínio de problemas considerados, passando a tratar questões temporais e também de recursos.

Portando, os elevados tempos retornados por este planejador, em alguns casos, não vem a ser necessariamente um problema. A possibilidade de futuramente tratar problemas não clássicos é um dos obejtivos de se investir no desenvolvimento de sistemas planejadores baseados na classe *petrinet*. Adicionalmente, a disponibilização de um método evolucionário para tratar as redes geradas por essa classe vem a ser uma contribuição muito importante, visto que não há, nos dias de hoje, um método eficiente capaz de tratar as mesmas.

ANEXO I

Descrição em PDDL dos problemas tratados pelo presente trabalho, organizados de acordo com o domínio ao qual cada um pertence (mundo dos blocos, gripper, mystery e logistics).

Mundo dos Blocos

```
(define (problem probex)
(:domain BLOCKS)
(:objects A B C - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ONTABLE C))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (ONTABLE A) (ONTABLE B)
       (ONTABLE C))))
```

```
(define (problem AnomaliaSussman)
(:domain BLOCKS)
(:objects B A C - block)
(:INIT  (CLEAR B) (ONTABLE B) (ONTABLE A) (ON C A) (CLEAR C))
(:goal (AND (ON C B) (ON B A))))
```

```
(define (problem prob01)
(:domain BLOCKS)
(:objects A B C D - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ONTABLE D))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (ONTABLE A)
       (ONTABLE B) (ONTABLE C) (ONTABLE D))))
```

```
(define (problem prob02)
(:domain BLOCKS)
(:objects A B C D E - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ONTABLE E))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (CLEAR E)
       (ONTABLE A) (ONTABLE B) (ONTABLE C) (ONTABLE D) (ONTABLE E))
))


(define (problem prob03)
(:domain BLOCKS)
(:objects A B C D E F - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F)
       (ONTABLE F))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (CLEAR E) (CLEAR F)
       (ONTABLE A) (ONTABLE B) (ONTABLE C) (ONTABLE D) (ONTABLE E)
       (ONTABLE F))))


(define (problem prob04)
(:domain BLOCKS)
(:objects A B C D E F G - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
       (ONTABLE G))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (CLEAR E) (CLEAR F)
       (CLEAR G) (ONTABLE A) (ONTABLE B) (ONTABLE C) (ONTABLE D)
       (ONTABLE E) (ONTABLE F) (ONTABLE G))))
```

```
(define (problem prob05)
(:domain BLOCKS)
(:objects A B C D E F G H - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F)
       (ON F G) (ON G H) (ONTABLE H))
(:goal (AND (CLEAR A) (CLEAR B) (CLEAR C) (CLEAR D) (CLEAR E)
       (CLEAR F) (CLEAR G) (CLEAR H) (ONTABLE A) (ONTABLE B)
       (ONTABLE C) (ONTABLE D) (ONTABLE E) (ONTABLE F) (ONTABLE G)
       (ONTABLE H))))

(define (problem probinverte05)
(:domain BLOCKS)
(:objects A B C D E - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ONTABLE E))
(:goal (AND (CLEAR E) (ON E D) (ON D C) (ON C B) (ON B A) (ONTABLE A))))


(define (problem probinverte06)
(:domain BLOCKS)
(:objects A B C D E F - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F)
       (ONTABLE F))
(:goal (AND (CLEAR F) (ON F E) (ON E D) (ON D C) (ON C B) (ON B A)
       (ONTABLE A))))


(define (problem probinverte07)
(:domain BLOCKS)
```

```
(:objects A B C D E F G - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
       (ONTABLE G))
(:goal (AND (CLEAR G) (ON G F) (ON F E) (ON E D) (ON D C) (ON C B)
       (ON B A) (ONTABLE A))))


(define (problem probinverte08)
(:domain BLOCKS)
(:objects A B C D E F G H - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
       (ON G H) (ONTABLE H))
(:goal (AND (CLEAR H) (ON H G) (ON G F) (ON F E) (ON E D) (ON D C)
       (ON C B) (ON B A) (ONTABLE A))))


(define (problem probinverte09)
(:domain BLOCKS)
(:objects A B C D E F G H I - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
       (ON G H) (ON H I) (ONTABLE I))
(:goal (AND (CLEAR I) (ON I H) (ON H G) (ON G F) (ON F E) (ON E D)
       (ON D C) (ON C B) (ON B A) (ONTABLE A))))


(define (problem probinverte10)
(:domain BLOCKS)
(:objects A B C D E F G H I J - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
```

```
       (ON G H) (ON H I) (ON I J) (ONTABLE J))
(:goal (AND (CLEAR J) (ON J I) (ON I H) (ON H G) (ON G F) (ON F E)
       (ON E D) (ON D C) (ON C B) (ON B A) (ONTABLE A))))
```

```
(define (problem probinverte11)
(:domain BLOCKS)
(:objects A B C D E F G H I J L - block)
(:INIT (CLEAR A) (ON A B) (ON B C) (ON C D) (ON D E) (ON E F) (ON F G)
       (ON G H) (ON H I) (ON I J) (ON J L) (ONTABLE L))
(:goal (AND (CLEAR L) (ON L J) (ON J I) (ON I H) (ON H G) (ON G F)
       (ON F E) (ON E D) (ON D C) (ON C B) (ON B A) (ONTABLE A))))
```

Gripper

```
(define (problem prob00)
   (:domain gripper-strips)
   (:objects rooma roomb - room
             ball2 ball1 - obj
             left right - gripper )
   (:init (free left) (free right) (at-robby rooma) (at ball2 rooma)
          (at ball1 rooma))
   (:goal (and (at ball2 roomb) (at ball1 roomb))))
```

```
(define (problem prob01)
   (:domain gripper-strips)
   (:objects rooma roomb - room
             ball4 ball3 ball2 ball1 - obj
```

```
            left right - gripper )
   (:init (free left) (free right) (at-robby rooma) (at ball4 rooma)
          (at ball3 rooma) (at ball2 rooma) (at ball1 rooma))
   (:goal (and (at ball4 roomb) (at ball3 roomb) (at ball2 roomb)
          (at ball1 roomb))))

(define (problem prob02)
   (:domain gripper-strips)
   (:objects rooma roomb - room
             ball6 ball5 ball4 ball3 ball2 ball1 - obj
             left right - gripper)
   (:init (at-robby rooma) (free left) (free right) (at ball6 rooma)
          (at ball5 rooma) (at ball4 rooma) (at ball3 rooma)
          (at ball2 rooma) (at ball1 rooma))
   (:goal (and (at ball6 roomb) (at ball5 roomb) (at ball4 roomb)
          (at ball3 roomb) (at ball2 roomb) (at ball1 roomb))))
```

Mystery

```
(define (problem prob00)
(:domain mystery-strips)
(:objects f0 f1 - fuel
          s0 s1 s2 - space
          l0 l1 - location
          v0 - vehicle
          c0 c1 - cargo)
(:init (fuel-neighbor f0 f1) (space-neighbor s0 s1)
       (space-neighbor s1 s2) (conn l0 l1) (conn l1 l0) (has-fuel l0 f1)
       (has-fuel l1 f0) (has-space  v0 s2) (at v0 l0) (at c0 l1)
```

```
       (at c1 l0))
(:goal (and (at c0 l1) (at c1 l1))))


(define (problem prob01)
(:domain mystery-strips)
(:objects f0 f1 f2 - fuel
          s0 s1 s2 - space
          l0 l1 - location
          v0 - vehicle
          c0 c1 - cargo)
(:init (fuel-neighbor f0 f1) (fuel-neighbor f1 f2) (space-neighbor s0 s1)
       (space-neighbor s1 s2) (conn l0 l1) (conn l1 l0) (has-fuel l0 f1)
       (has-fuel l1 f2) (has-space  v0 s2) (at v0 l1) (at c0 l1)
       (at c1 l0))
(:goal (and (at c0 l0) (at c1 l1) )))


(define (problem prob02)
(:domain mystery-strips)
(:objects f0 f1 f2 - fuel
          s0 s1 s2 - space
          l0 l1 l2 - location
          v0 v1 - vehicle
          c0 c1 c2 c3 c4 - cargo)
(:init (fuel-neighbor f0 f1) (fuel-neighbor f1 f2) (space-neighbor s0 s1)
       (space-neighbor s1 s2) (conn l0 l1) (conn l1 l0) (conn l1 l2)
       (conn l2 l1) (conn l2 l0) (conn l0 l2) (has-fuel l0 f1)
       (has-fuel l1 f2) (has-fuel l2 f1) (has-space  v0 s2)
```

```
       (has-space  v1 s2) (at v0 l1) (at v1 l1) (at c0 l0) (at c1 l0)
       (at c2 l1) (at c3 l1) (at c4 l2))
(:goal (and (at c0 l2) (at c1 l2) (at c2 l0) (at c3 l2) (at c4 l1))))
```

Logistics

```
(define (problem prob001)
   (:domain logistics-strips)
   (:objects package1 - package
             truck1 - truck
             city2 city1 - location)
   (:init (at truck1 city2) (at package1 city1))
   (:goal (and (at package1 city2) )))


(define (problem prob002)
   (:domain logistics-strips)
   (:objects package1 package2 - package
             truck1 - truck
             city2 city1 - location)
   (:init (at truck1 city1) (at package1 city1) (at package2 city1))
   (:goal (and (at package1 city2) (at package2 city2) )))


(define (problem prob003)
   (:domain logistics-strips)
   (:objects package1 package2 - package
             truck1 - truck
             city2 city1 - location)
   (:init (at truck1 city1) (at package1 city1) (at package2 city2))
```

```
   (:goal (and (at package1 city2) (at package2 city1) )))


(define (problem prob004)
   (:domain logistics-strips)
   (:objects package1 package2 - package
             truck1 truck2 - truck
             city3 city2 city1 - location)
   (:init (at truck1 city1) (at truck2 city2) (at package1 city1)
          (at package2 city2))
   (:goal (and (at package1 city3) (at package2 city3) (at truck1 city1)
          (at truck2 city2))))


(define (problem prob005)
   (:domain logistics-strips)
   (:objects package1 package2 package3 package4 - package
             truck1 truck2 - truck
             city2 city1 - location)
   (:init (at truck1 city1) (at package1 city1) (at package2 city1)
          (at package3 city1) (at package4 city1))
   (:goal (and (at package1 city2) (at package2 city2)
          (at package3 city2) (at package4 city2) )))


(define (problem prob006)
   (:domain logistics-strips)
   (:objects package1 package2 package3 package4 - package
             truck1 truck2 truck3 - truck
```

```
            city1 city2 city3 - location)
   (:init (at truck1 city1) (at truck2 city2) (at truck3 city3)
          (at package1 city1) (at package2 city2) (at package3 city3)
          (at package4 city3))
   (:goal (and (at package1 city2) (at package2 city3)
          (at package3 city1) (at package4 city1) )))

(define (problem prob007)
   (:domain logistics-strips)
   (:objects package1 package2 package3 package4 - package
             truck1 truck2 - truck
             city1 city2 city3 - location)
   (:init (at truck1 city1) (at truck2 city2) (at package1 city1)
          (at package2 city1) (at package3 city2) (at package4 city2))
   (:goal (and (at package1 city2) (at package2 city3)
          (at package3 city1) (at package4 city3) )))


(define (problem prob008)
   (:domain logistics-strips)
   (:objects package1 package2 package3 package4 package5 - package
             truck1 truck2 - truck
             city1 city2 city3 - location)
   (:init (at truck1 city1) (at truck2 city2) (at package1 city1)
          (at package2 city1) (at package3 city2) (at package4 city2)
          (at package5 city3))
   (:goal (and (at package1 city2) (at package2 city3)
          (at package3 city1) (at package4 city3) (at package5 city1))))
```

BIBLIOGRAFIA

[1] D. S. Weld. Recent advances in ai planning. *AI Magazine*, 20(2):93–123, 1999.

[2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[3] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence*, 2:3–4, 1971.

[4] C. C. Green. Application of theorem proving to problem solving. In D. E. Walker and L. M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence, IJCAI*, pages 219–240. Morgan Kaufmann, 1969.

[5] B. Raphael. The frame problem in problem-solving systems. In N.V. Findler and B. Meltzer, editors, *Proceedings of the Advanced Study Institute on Artificial Intelligence and Heuristic Programming*, pages 159–169, Edinburgh, UK, 1971. Edinburgh University Press.

[6] Bertram Raphael. The frame problem in problem-solving systems. Technical Report 33, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, June 1970. Paper prepared for publication in the Proceedings of the Advanced Study Institute on Artificial Intelligence and Heuristic Programming, held at Menaggio, Italy, August 1970.

[7] T. Bylander. The computational complexity of propositional strips planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[8] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

[9] J. S. Penberthy and D. S. Weld. Ucpop: A sound, complete, partial order planner for adl. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, pages 103–114. Kaufmann, San Mateo, CA, 1992.

[10] H. Kautz and B. Selman. Planning as satisfiability. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 359–363, New York, NY, USA, 1992. John Wiley & Sons, Inc.

[11] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. In *Proc. of the 14th IJCAI*, pages 1636–1642, Montreal, Canada, 1995.

[12] H. A. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *IJCAI*, pages 318–325, 1999.

[13] J. Hoffmann and B. Nebel. The ff planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[14] F. Silva, M.A. Castilho, and L.A. Künzle. Petriplan: a new algorithm for plan generation (preliminary report). In *Lecture Notes in Artificial Intelligence*, volume 1952, pages 86–95, sep 2000. International Joint Conference IBERAMIA'2000 - SBIA'2000.

[15] F. Silva. Algoritmos para planificação baseados em strips. Master's thesis, Universidade Federal do Paraná, Curitiba PR Brasil, outubro 2000.

[16] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Abril 1989.

[17] R. A. N. R. Montaño. Aplicação de formas não-clausais em planejamento com redes de petri. Master's thesis, Universidade Federal do Paraná, Curitiba PR Brasil, outubro 2006.

[18] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[19] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[20] E. Lecheta. Algoritmos genéticos para planejamento em inteligência artificial. Master's thesis, Universidade Federal do Paraná, Curitiba PR Brasil, fevereiro 2004.

[21] M. A. Castilho, L. A. Künzle, E. Lecheta, V. Palodeto, and F. Silva. An investigation on genetic algorithms for generic strips planning. In *IBERAMIA*, pages 185–194, 2004.

[22] C. Westerberg and J. Levine. Genplan: Combining genetic programming and planning. In Proc. of the 19th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2000)., 2000.

[23] F. Lardeux, F. Saubion, and Jin-Kao Hao. The gasat solver. In *Proc. of the SAT 2004 Competition : Solver Descriptions*, Vancouver, Canada, may 2004.

[24] F. Lardeux, F. Saubion, and Jin-Kao Hao. Gasat: a genetic local search algorithm for the satisfiability problem. *Evol. Comput.*, 14(2):223–253, 2006.

[25] R. Reiter. On closed world data bases. Technical report, Vancouver, BC, Canada, Canada, 1977.

[26] D. V. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[27] D. McDermott. Pddl — the planning domain definition language, 1998.

[28] E. P. D. Pednault. Adl: exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[29] D. E. Wilkins. *Practical planning: extending the classical AI planning paradigm*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[30] K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[31] K. Erol, J. A. Hendler, and D. S. Nau. Umcp: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254, 1994.

[32] J. E. Marynowski. Ambiente de planejamento ipê. Master's thesis, Universidade Federal do Paraná, Curitiba PR, Brasil, novembro 2004.

[33] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.

[34] Charles Darwin. *On The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, London, 1859.

[35] G. J. Sussman. *A Computer Model of Skill Acquisition.* American Elsevier, New York, 1975.